SilverLining

Generated by Doxygen 1.7.3

Fri Oct 16 2015 14:39:58

Contents

1 SilverLining(tm) Software Development Kit (SDK) User's Manual **1**

2 SilverLining Licensing **5**

3 SilverLining Support **7**

4 Copyright Notices **9**

5 SilverLining Troubleshooting Tips **11**

5.1 "The application has failed to start because the application's configuration is incorrect." . . . 11
5.2 SilverLining crashes or hangs at startup . . . 12
5.3 Atmosphere::Initialize() returns an error code . . . 12
5.4 My sky is sideways / upside down / otherwise misoriented . . . 13
5.5 My sky doesn't appear or doesn't move with the camera . . . 13
5.6 SilverLining slows down my framerate too much . . . 14
5.7 My framerate gets slower over time when I'm using SilverLining, for no apparent reason . . . 14
5.8 My sky or clouds are rendering funny . . . 15
5.9 I use an unusual coordinate system, and the clouds aren't showing up where I expect them . . . 15
5.10 SilverLining draws to all active render targets, but I only want it to draw to specific ones. . . . 15
5.11 My problem isn't listed here! . . . 16

6 Overview of SilverLining **17**

6.1 SDK Contents . . . 17
6.1.1 Windows Build Settings . . . 18
6.1.2 Linux Build Settings and Dependencies . . . 19
6.1.3 Building with C# . . . 20
6.2 Distributing SilverLining with Your Application . . . 20
6.3 Overview of SilverLining . . . 21

7 Using SilverLining **23**

7.1 A simple example . . . 23
7.1.1 Initialization . . . 23

7.1.2 Infinite Cloud Layers . . . 26
7.1.3 Cloud Animation Effects . . . 27
7.1.4 Integrating SilverLining with your Rendering Loop . . . 27
7.1.5 Lighting Your Scene with SilverLining . . . 30
7.1.6 Fog Effects with SilverLining . . . 31
7.2 Drawing Your Skybox Last Instead of First . . . 34
7.3 Integration with Multi-Threaded Renderers . . . 35
7.4 Memory Management with SilverLining . . . 35
7.5 Resource Management with SilverLining . . . 36
7.6 Stratocumulus Cloud Layers . . . 36
7.7 Simulating Sand Storms . . . 37
7.8 Integration Tips with Popular Scene Graphs . . . 37
7.8.1 Integrating with OpenSceneGraph . . . 37
7.8.2 Integrating with Ogre3D . . . 39
7.8.3 Integrating with Gamebryo(R) Lightspeed(tm) . . . 39
7.8.4 Integrating with Carmenta Engine . . . 39
7.8.5 Integrating with NVidia SceniX . . . 39
7.9 Using SilverLining in Geocentric or ECEF (Earth Centered Earth Fixed) Coordinates . . . 40
7.10 Simulating Precipitation with SilverLining . . . 41
7.11 Manipulating Time with SilverLining . . . 42
7.12 Considerations for Multi-Channel, Multi-Viewport, and Multi-Context Setups . . . 43
7.13 Managing Your Own Drawing . . . 44
7.14 Crepuscular Rays . . . 46
7.15 Advanced: using shadow maps . . . 46
7.16 Creating environment maps from SilverLining . . . 48
7.17 Using SilverLining with High Dynamic Range applications (HDR) . . . 48
7.18 Selecting a Sky Model . . . 49
7.19 Positioning Individual Clouds Within a Cloud Layer . . . 50
7.20 Using SilverLining with Linear Color Space . . . 51
7.21 Tweaking SilverLining's Appearance . . . 51

8 Class Index **53**
8.1 Class Hierarchy . . . 53

9 Class Index **55**
9.1 Class List . . . 55

10 File Index **57**
10.1 File List . . . 57

11 Class Documentation **59**
11.1 SilverLining::Allocator Class Reference . . . 59
11.1.1 Detailed Description . . . 60
11.1.2 Member Function Documentation . . . 60

11.1.2.1 SetAllocator 60
11.2 SilverLining::Atmosphere Class Reference 60
11.2.1 Detailed Description 71
11.2.2 Constructor & Destructor Documentation 72
11.2.2.1 Atmosphere 72
11.2.2.2 ~Atmosphere 72
11.2.3 Member Function Documentation 72
11.2.3.1 ClearCloudLightingOverride 72
11.2.3.2 ClearObjects 72
11.2.3.3 CullObjects 73
11.2.3.4 DisableFarCulling 73
11.2.3.5 DrawObject 73
11.2.3.6 DrawObjects 74
11.2.3.7 DrawSky 76
11.2.3.8 EnableHDR 77
11.2.3.9 EnableLensFlare 78
11.2.3.10 EnableSunOcclusion 78
11.2.3.11 ForceLightingRecompute 78
11.2.3.12 ForceMoonPhase 78
11.2.3.13 ForceSunAndMoon 78
11.2.3.14 ForceSunAndMoon 79
11.2.3.15 ForceSunAndMoonHorizon 79
11.2.3.16 ForceSunAndMoonHorizon 79
11.2.3.17 GetAmbientColor 79
11.2.3.18 GetBillboardShader 80
11.2.3.19 GetCameraMatrix 80
11.2.3.20 GetCloudBounds 80
11.2.3.21 GetConditions 80
11.2.3.22 GetConditions 81
11.2.3.23 GetConfigOptionBoolean 81
11.2.3.24 GetConfigOptionDouble 81
11.2.3.25 GetConfigOptionString 81
11.2.3.26 GetEnvironmentMap 81
11.2.3.27 GetFarCullingDisabled 82
11.2.3.28 GetFogEnabled 82
11.2.3.29 GetFogSettings 83
11.2.3.30 GetFramerate 83
11.2.3.31 GetGamma 83
11.2.3.32 GetHaze 83
11.2.3.33 GetHDREnabled 84
11.2.3.34 GetHorizonColor 84
11.2.3.35 GetHorizonColor 84
11.2.3.36 GetInfraRedMode 85
11.2.3.37 GetLensFlareEnabled 85
11.2.3.38 GetMoonAlpha 85
11.2.3.39 GetMoonColor 85

11.2.3.40 GetMoonPosition . . . 86
11.2.3.41 GetMoonPositionEquatorial . . . 86
11.2.3.42 GetMoonPositionGeographic . . . 86
11.2.3.43 GetObjectDistance . . . 87
11.2.3.44 GetObjectDistance . . . 87
11.2.3.45 GetObjectPosition . . . 88
11.2.3.46 GetOutputScale . . . 88
11.2.3.47 GetPrecipitationShader . . . 88
11.2.3.48 GetProjectionMatrix . . . 88
11.2.3.49 GetResourceLoader . . . 89
11.2.3.50 GetRightVector . . . 89
11.2.3.51 GetShadowMap . . . 89
11.2.3.52 GetSkyCoverage . . . 91
11.2.3.53 GetSkyModel . . . 91
11.2.3.54 GetSkyShader . . . 91
11.2.3.55 GetStarShader . . . 91
11.2.3.56 GetSunAlpha . . . 91
11.2.3.57 GetSunColor . . . 91
11.2.3.58 GetSunOcclusion . . . 92
11.2.3.59 GetSunOcclusionEnabled . . . 92
11.2.3.60 GetSunOrMoonColor . . . 92
11.2.3.61 GetSunOrMoonPosition . . . 93
11.2.3.62 GetSunOrMoonPositionEquatorial . . . 93
11.2.3.63 GetSunOrMoonPositionGeographic . . . 93
11.2.3.64 GetSunPosition . . . 93
11.2.3.65 GetSunPositionEquatorial . . . 94
11.2.3.66 GetSunPositionGeographic . . . 94
11.2.3.67 GetUnitScale . . . 94
11.2.3.68 GetUpVector . . . 94
11.2.3.69 GetViewport . . . 94
11.2.3.70 GetZenithColor . . . 94
11.2.3.71 Initialize . . . 95
11.2.3.72 Initialize . . . 96
11.2.3.73 IsInitialized . . . 96
11.2.3.74 OverrideCloudLighting . . . 96
11.2.3.75 ReloadConfigFile . . . 97
11.2.3.76 ReloadCumulusTextures . . . 97
11.2.3.77 ReloadShaders . . . 97
11.2.3.78 Serialize . . . 97
11.2.3.79 SetCameraMatrix . . . 98
11.2.3.80 SetConditions . . . 98
11.2.3.81 SetConfigOption . . . 98
11.2.3.82 SetDepthRange . . . 98
11.2.3.83 SetGamma . . . 99
11.2.3.84 SetHaze . . . 99
11.2.3.85 SetInfraRedMode . . . 99

11.2.3.86 SetMoonAlpha . . . 99
11.2.3.87 SetOutputScale . . . 100
11.2.3.88 SetProjectionMatrix . . . 100
11.2.3.89 SetRandomNumberGenerator . . . 100
11.2.3.90 SetResourceLoader . . . 100
11.2.3.91 SetRightVector . . . 101
11.2.3.92 SetSkyModel . . . 101
11.2.3.93 SetSunAlpha . . . 101
11.2.3.94 SetUpVector . . . 102
11.2.3.95 SetViewport . . . 102
11.2.3.96 SetWorldUnits . . . 102
11.2.3.97 SL_VECTOR . . . 103
11.2.3.98 SL_VECTOR . . . 103
11.2.3.99 Unserialize . . . 103
11.2.3.100 UpdateEphemeris . . . 103
11.2.3.101 UpdateSkyAndClouds . . . 103
11.3 SilverLining::AtmosphericConditions Class Reference . . . 104
11.3.1 Detailed Description . . . 108
11.3.2 Constructor & Destructor Documentation . . . 108
11.3.2.1 AtmosphericConditions . . . 108
11.3.2.2 ~AtmosphericConditions . . . 108
11.3.3 Member Function Documentation . . . 109
11.3.3.1 AddCloudLayer . . . 109
11.3.3.2 ApplyWind . . . 109
11.3.3.3 ClearFog . . . 109
11.3.3.4 ClearWindVolumes . . . 109
11.3.3.5 EnableTimePassage . . . 110
11.3.3.6 GetCloudLayer . . . 110
11.3.3.7 GetFog . . . 111
11.3.3.8 GetLightPollution . . . 111
11.3.3.9 GetLocation . . . 111
11.3.3.10 GetMillisecondTimer . . . 111
11.3.3.11 GetPrecipitationWind . . . 111
11.3.3.12 GetTime . . . 112
11.3.3.13 GetTurbidity . . . 112
11.3.3.14 GetVisibility . . . 112
11.3.3.15 GetWind . . . 112
11.3.3.16 RemoveAllCloudLayers . . . 113
11.3.3.17 RemoveCloudLayer . . . 113
11.3.3.18 RemoveWindVolume . . . 113
11.3.3.19 Serialize . . . 113
11.3.3.20 SetFog . . . 113
11.3.3.21 SetLightPollution . . . 114
11.3.3.22 SetLocation . . . 114
11.3.3.23 SetMillisecondTimer . . . 114
11.3.3.24 SetPrecipitation . . . 115

11.3.3.25 SetPrecipitationWind . . . 116
11.3.3.26 SetPresetConditions . . . 116
11.3.3.27 SetTime . . . 116
11.3.3.28 SetTurbidity . . . 117
11.3.3.29 SetVisibility . . . 117
11.3.3.30 SetWind . . . 117
11.3.3.31 SL_MAP . . . 118
11.3.3.32 SL_MAP . . . 118
11.4 SilverLining::CloudLayer Class Reference . . . 118
11.4.1 Detailed Description . . . 124
11.4.2 Constructor & Destructor Documentation . . . 125
11.4.2.1 CloudLayer . . . 125
11.4.2.2 ~CloudLayer . . . 125
11.4.3 Member Function Documentation . . . 125
11.4.3.1 AddCloudAt . . . 125
11.4.3.2 AddLightningListener . . . 127
11.4.3.3 ClearCloudColorOverride . . . 127
11.4.3.4 ClearClouds . . . 127
11.4.3.5 ClearLightningListeners . . . 127
11.4.3.6 ExportToVRML . . . 127
11.4.3.7 ForceLightning . . . 128
11.4.3.8 GetAlpha . . . 128
11.4.3.9 GetBaseAltitude . . . 128
11.4.3.10 GetBaseLength . . . 128
11.4.3.11 GetBaseWidth . . . 128
11.4.3.12 GetCloudWrapping . . . 129
11.4.3.13 GetCurveTowardGround . . . 129
11.4.3.14 GetDensity . . . 129
11.4.3.15 GetDensityMultiplier . . . 129
11.4.3.16 GetEnabled . . . 129
11.4.3.17 GetFadeTowardEdges . . . 130
11.4.3.18 GetIsInfinite . . . 130
11.4.3.19 GetLayerPosition . . . 130
11.4.3.20 GetLightningDischargeMode . . . 130
11.4.3.21 GetMaxHeight . . . 130
11.4.3.22 GetOverriddenCloudColor . . . 131
11.4.3.23 GetThickness . . . 131
11.4.3.24 GetType . . . 131
11.4.3.25 GetWind . . . 131
11.4.3.26 HasPrecipitationAtPosition . . . 131
11.4.3.27 Intersect . . . 132
11.4.3.28 IsInsideCloud . . . 132
11.4.3.29 IsRenderable . . . 132
11.4.3.30 OverrideCloudColor . . . 132
11.4.3.31 Restore . . . 133
11.4.3.32 Save . . . 133

11.4.3.33 SeedClouds . . . 134
11.4.3.34 Serialize . . . 134
11.4.3.35 SetAlpha . . . 135
11.4.3.36 SetBaseAltitude . . . 135
11.4.3.37 SetBaseLength . . . 135
11.4.3.38 SetBaseWidth . . . 135
11.4.3.39 SetCloudAnimationEffects . . . 136
11.4.3.40 SetCloudWrapping . . . 137
11.4.3.41 SetCurveTowardGround . . . 137
11.4.3.42 SetDensity . . . 138
11.4.3.43 SetDensityMultiplier . . . 138
11.4.3.44 SetEnabled . . . 138
11.4.3.45 SetFadeTowardEdges . . . 139
11.4.3.46 SetIsInfinite . . . 139
11.4.3.47 SetLayerPosition . . . 140
11.4.3.48 SetLightningDischargeMode . . . 141
11.4.3.49 SetPrecipitation . . . 141
11.4.3.50 SetThickness . . . 142
11.4.3.51 SetType . . . 142
11.4.3.52 SetWind . . . 143
11.4.3.53 SL_MAP . . . 143
11.4.3.54 SupportsAddCloudAt . . . 143
11.5 SilverLining::CloudLayerFactory Class Reference . . . 143
11.5.1 Detailed Description . . . 145
11.5.2 Member Function Documentation . . . 145
11.5.2.1 Create . . . 145
11.5.2.2 Serialize . . . 146
11.6 SilverLining::Color Class Reference . . . 146
11.6.1 Detailed Description . . . 148
11.6.2 Constructor & Destructor Documentation . . . 149
11.6.2.1 Color . . . 149
11.6.2.2 Color . . . 149
11.6.2.3 Color . . . 149
11.6.2.4 Color . . . 149
11.6.2.5 Color . . . 149
11.6.2.6 Color . . . 149
11.6.2.7 Color . . . 149
11.6.3 Member Function Documentation . . . 150
11.6.3.1 ClampToUnitOrLess . . . 150
11.6.3.2 operator* . . . 150
11.6.3.3 operator* . . . 150
11.6.3.4 operator+ . . . 150
11.6.3.5 ScaleToUnitOrLess . . . 150
11.6.3.6 ScaleToValueOrLess . . . 150
11.6.3.7 Serialize . . . 150
11.6.3.8 ToGrayscale . . . 150

11.6.3.9 Unserialize . . . 151
11.7 SilverLining::Frustum Class Reference . . . 151
11.7.1 Detailed Description . . . 153
11.7.2 Member Enumeration Documentation . . . 153
11.7.2.1 Planes . . . 153
11.7.3 Constructor & Destructor Documentation . . . 153
11.7.3.1 Frustum . . . 153
11.7.3.2 ~Frustum . . . 153
11.7.4 Member Function Documentation . . . 153
11.7.4.1 GetPlane . . . 153
11.7.4.2 SetPlane . . . 154
11.8 SilverLining::LightningListener Class Reference . . . 154
11.8.1 Detailed Description . . . 154
11.8.2 Constructor & Destructor Documentation . . . 155
11.8.2.1 ~LightningListener . . . 155
11.8.3 Member Function Documentation . . . 155
11.8.3.1 LightningStartedEvent . . . 155
11.9 SilverLining::LocalTime Class Reference . . . 155
11.9.1 Detailed Description . . . 158
11.9.2 Constructor & Destructor Documentation . . . 159
11.9.2.1 LocalTime . . . 159
11.9.2.2 ~LocalTime . . . 159
11.9.3 Member Function Documentation . . . 159
11.9.3.1 AddSeconds . . . 159
11.9.3.2 GetDay . . . 159
11.9.3.3 GetEpoch1990Days . . . 159
11.9.3.4 GetEpoch2000Centuries . . . 160
11.9.3.5 GetHour . . . 160
11.9.3.6 GetJulianDate . . . 160
11.9.3.7 GetMinutes . . . 161
11.9.3.8 GetMonth . . . 161
11.9.3.9 GetObservingDaylightSavingsTime . . . 161
11.9.3.10 GetSeconds . . . 161
11.9.3.11 GetTimeZone . . . 162
11.9.3.12 GetYear . . . 162
11.9.3.13 Serialize . . . 162
11.9.3.14 SetDay . . . 162
11.9.3.15 SetFromEpochSeconds . . . 162
11.9.3.16 SetFromSystemTime . . . 163
11.9.3.17 SetHour . . . 163
11.9.3.18 SetMinutes . . . 163
11.9.3.19 SetMonth . . . 163
11.9.3.20 SetObservingDaylightSavingsTime . . . 164
11.9.3.21 SetSeconds . . . 164
11.9.3.22 SetTimeZone . . . 164
11.9.3.23 SetYear . . . 165

11.10SilverLining::Location Class Reference 165
11.10.1 Detailed Description 167
11.10.2 Constructor & Destructor Documentation 167
11.10.2.1 Location 167
11.10.3 Member Function Documentation 167
11.10.3.1 GetAltitude 167
11.10.3.2 GetLatitude 167
11.10.3.3 GetLongitude 168
11.10.3.4 Serialize 168
11.10.3.5 SetAltitude 168
11.10.3.6 SetLatitude 168
11.10.3.7 SetLongitude 169
11.11SilverLining::Matrix3 Class Reference 169
11.11.1 Detailed Description 171
11.11.2 Constructor & Destructor Documentation 171
11.11.2.1 Matrix3 171
11.11.2.2 Matrix3 172
11.11.2.3 ∼Matrix3 172
11.11.3 Member Function Documentation 172
11.11.3.1 FromAxisAngle 172
11.11.3.2 FromRx 172
11.11.3.3 FromRy 172
11.11.3.4 FromRz 172
11.11.3.5 FromXYZ 172
11.11.3.6 Inverse 172
11.11.3.7 operator∗ 173
11.11.3.8 operator∗ 173
11.11.3.9 Transpose 173
11.11.4 Friends And Related Function Documentation 173
11.11.4.1 operator∗ 173
11.12SilverLining::Matrix4 Class Reference 173
11.12.1 Detailed Description 176
11.12.2 Constructor & Destructor Documentation 176
11.12.2.1 Matrix4 176
11.12.2.2 Matrix4 176
11.12.2.3 Matrix4 176
11.12.2.4 Matrix4 176
11.12.2.5 ∼Matrix4 176
11.12.3 Member Function Documentation 176
11.12.3.1 GetRow 176
11.12.3.2 operator∗ 176
11.12.3.3 operator∗ 177
11.12.3.4 operator∗ 177
11.12.3.5 Transpose 177
11.12.4 Friends And Related Function Documentation 177
11.12.4.1 operator∗ 177

11.13 SilverLining::MemObject Class Reference . . . 177
11.13.1 Detailed Description . . . 179
11.14 SilverLining::MillisecondTimer Class Reference . . . 179
11.14.1 Detailed Description . . . 179
11.14.2 Constructor & Destructor Documentation . . . 180
11.14.2.1 MillisecondTimer . . . 180
11.14.2.2 ~MillisecondTimer . . . 180
11.14.3 Member Function Documentation . . . 180
11.14.3.1 GetMilliseconds . . . 180
11.15 SilverLining::Plane Class Reference . . . 180
11.15.1 Detailed Description . . . 182
11.15.2 Constructor & Destructor Documentation . . . 182
11.15.2.1 Plane . . . 182
11.15.2.2 Plane . . . 182
11.15.3 Member Function Documentation . . . 183
11.15.3.1 GetDistance . . . 183
11.15.3.2 GetNormal . . . 183
11.15.3.3 Normalize . . . 183
11.15.3.4 SetDistance . . . 183
11.15.3.5 SetNormal . . . 183
11.16 SilverLining::RandomNumberGenerator Class Reference . . . 183
11.16.1 Detailed Description . . . 184
11.16.2 Constructor & Destructor Documentation . . . 184
11.16.2.1 ~RandomNumberGenerator . . . 184
11.17 SilverLining::ResourceLoader Class Reference . . . 184
11.17.1 Detailed Description . . . 185
11.17.2 Member Function Documentation . . . 186
11.17.2.1 FreeResource . . . 186
11.17.2.2 GetFilesInDirectory . . . 186
11.17.2.3 LoadResource . . . 186
11.17.2.4 SetResourceDirPath . . . 187
11.18 SilverLining::Vector3 Class Reference . . . 187
11.18.1 Detailed Description . . . 189
11.18.2 Constructor & Destructor Documentation . . . 190
11.18.2.1 Vector3 . . . 190
11.18.2.2 Vector3 . . . 190
11.18.3 Member Function Documentation . . . 190
11.18.3.1 Cross . . . 190
11.18.3.2 Dot . . . 190
11.18.3.3 Length . . . 190
11.18.3.4 Normalize . . . 190
11.18.3.5 operator!= . . . 190
11.18.3.6 operator* . . . 190
11.18.3.7 operator* . . . 191
11.18.3.8 operator+ . . . 191
11.18.3.9 operator+ . . . 191

11.18.3.10operator- . . . 191
11.18.3.1 loperator== . . . 191
11.18.3.12Serialize . . . 191
11.18.3.13SquaredLength . . . 191
11.18.3.14Unserialize . . . 191
11.18.4 Member Data Documentation . . . 192
11.18.4.1 x . . . 192
11.19SilverLining::Vector3f Class Reference . . . 192
11.19.1 Detailed Description . . . 194
11.19.2 Constructor & Destructor Documentation . . . 194
11.19.2.1 Vector3f . . . 194
11.19.2.2 Vector3f . . . 194
11.19.2.3 Vector3f . . . 194
11.19.3 Member Function Documentation . . . 194
11.19.3.1 Dot . . . 194
11.19.3.2 Length . . . 195
11.19.3.3 Normalize . . . 195
11.19.3.4 operator∗ . . . 195
11.19.3.5 operator+ . . . 195
11.19.3.6 operator- . . . 195
11.19.4 Member Data Documentation . . . 195
11.19.4.1 x . . . 195
11.20SilverLining::Vector4 Class Reference . . . 195
11.20.1 Detailed Description . . . 197
11.20.2 Constructor & Destructor Documentation . . . 197
11.20.2.1 Vector4 . . . 197
11.20.2.2 Vector4 . . . 197
11.20.3 Member Function Documentation . . . 197
11.20.3.1 Dot . . . 197
11.20.3.2 operator∗ . . . 198
11.20.4 Member Data Documentation . . . 198
11.20.4.1 x . . . 198
11.21SilverLining::Vertex Class Reference . . . 198
11.21.1 Detailed Description . . . 199
11.21.2 Member Function Documentation . . . 200
11.21.2.1 SetColor . . . 200
11.21.2.2 SetColor . . . 200
11.21.2.3 SetUV . . . 200
11.22SilverLining::WindVolume Class Reference . . . 200
11.22.1 Detailed Description . . . 202
11.22.2 Constructor & Destructor Documentation . . . 203
11.22.2.1 WindVolume . . . 203
11.22.2.2 ∼WindVolume . . . 203
11.22.3 Member Function Documentation . . . 203
11.22.3.1 GetDirection . . . 203
11.22.3.2 GetMaxAltitude . . . 203

11.22.3.3 GetMinAltitude . . . 203
11.22.3.4 GetWindSpeed . . . 204
11.22.3.5 Inside . . . 204
11.22.3.6 Serialize . . . 204
11.22.3.7 SetDirection . . . 204
11.22.3.8 SetMaxAltitude . . . 204
11.22.3.9 SetMinAltitude . . . 205
11.22.3.10 SetWindSpeed . . . 205

12 File Documentation **207**
12.1 C:/sl/Public Headers/Atmosphere.h File Reference . . . 207
12.1.1 Detailed Description . . . 208
12.2 C:/sl/Public Headers/AtmosphericConditions.h File Reference . . . 209
12.2.1 Detailed Description . . . 210
12.3 C:/sl/Public Headers/CloudLayer.h File Reference . . . 210
12.3.1 Detailed Description . . . 211
12.4 C:/sl/Public Headers/CloudLayerFactory.h File Reference . . . 211
12.4.1 Detailed Description . . . 212
12.5 C:/sl/Public Headers/CloudTypes.h File Reference . . . 212
12.5.1 Detailed Description . . . 213
12.5.2 Enumeration Type Documentation . . . 213
12.5.2.1 CloudTypes . . . 213
12.6 C:/sl/Public Headers/Color.h File Reference . . . 214
12.6.1 Detailed Description . . . 215
12.7 C:/sl/Public Headers/Frustum.h File Reference . . . 215
12.7.1 Detailed Description . . . 217
12.8 C:/sl/Public Headers/LightningListener.h File Reference . . . 217
12.8.1 Detailed Description . . . 218
12.9 C:/sl/Public Headers/LocalTime.h File Reference . . . 218
12.9.1 Detailed Description . . . 220
12.9.2 Enumeration Type Documentation . . . 220
12.9.2.1 TimeZones . . . 220
12.10 C:/sl/Public Headers/Location.h File Reference . . . 221
12.10.1 Detailed Description . . . 223
12.11 C:/sl/Public Headers/Matrix3.h File Reference . . . 223
12.11.1 Detailed Description . . . 224
12.12 C:/sl/Public Headers/Matrix4.h File Reference . . . 224
12.12.1 Detailed Description . . . 226
12.13 C:/sl/Public Headers/MemAlloc.h File Reference . . . 226
12.13.1 Detailed Description . . . 227
12.14 C:/sl/Public Headers/MillisecondTimer.h File Reference . . . 228
12.14.1 Detailed Description . . . 229
12.15 C:/sl/Public Headers/Plane.h File Reference . . . 229
12.15.1 Detailed Description . . . 231
12.16 C:/sl/Public Headers/RandomNumberGenerator.h File Reference . . . 231
12.16.1 Detailed Description . . . 233

12.17C:/sl/Public Headers/ResourceLoader.h File Reference 233
12.17.1 Detailed Description 234
12.18C:/sl/Public Headers/SilverLiningDLLCommon.h File Reference . . . 234
12.18.1 Detailed Description 235
12.19C:/sl/Public Headers/Vector3.h File Reference 236
12.19.1 Detailed Description 237
12.20C:/sl/Public Headers/Vector4.h File Reference 237
12.20.1 Detailed Description 239
12.21C:/sl/Public Headers/WindVolume.h File Reference 239
12.21.1 Detailed Description 241

Chapter 1

SilverLining(tm) Software Development Kit (SDK) User's Manual

Thanks for using SilverLining! SilverLining is a class library for real-time visual simulation of the atmosphere, clouds, sky, precipitation, sun, moon, stars, and planets. SilverLining allows you to create highly realistic, high-performance outdoor scenes with 3D clouds for any location at any time of day.

Although SilverLining includes some very complex physical simulations of extraterrestrial light passing through the atmosphere and clouds, using it within your application is very simple. All you need to do is initialize SilverLining to tell it some information about the outdoor conditions you wish to simulate, and then call a single method at the beginning of your frame rendering, and another at the end.

SilverLining is designed to easily plug into any renderer written on top of OpenGL 2.0 and above, OpenGL 3.2 "core contexts", DirectX 9, DirectX 10, or DirectX 11. Integration code is also provided for OpenSceneGraph, SceniX, Carmenta, C#, XNA, and Ogre3D, and is available separately for Unity and for Gamebryo(R) Lightspeed(tm).

All you need is the ability to call SilverLining when your frame begins, and again when your frame is over and you're ready to draw translucent objects in the scene. Users who require lower-level access also have the underlying cloud objects available so you can draw them individually within your scene graph, and licensed customers receive full source code if even lower-level access is required. We provide libraries for Windows, MacOS, and Linux developers, and customers have successfully ported SilverLining to the XBox360. A C# wrapper is also provided on top of our C++ libraries, and iOS and Android versions of SilverLining are available separately from our website.

This documentation will get you started. In addition to the class reference information included here, some additional information is also available:

- SilverLining Licensing How to obtain a license for SilverLining.

- Overview of SilverLining A high-level overview of SilverLining, its components, and its classes.
  - SDK Contents
  - Distributing SilverLining with Your Application
  - Windows Build Settings
  - Linux Build Settings and Dependencies
  - Building with C#

- Using SilverLining Step-by-step examples of how to integrate SilverLining into your application.
  - A simple example
    - Initialization
    - Infinite Cloud Layers
    - Cloud Animation Effects
    - Integrating SilverLining with your Rendering Loop
    - Lighting Your Scene with SilverLining
    - Fog Effects with SilverLining
  - Drawing Your Skybox Last Instead of First
  - Integration with Multi-Threaded Renderers
  - Memory Management with SilverLining
  - Resource Management with SilverLining
  - Stratocumulus Cloud Layers
  - Simulating Sand Storms
  - Integration Tips with Popular Scene Graphs
  - Using SilverLining in Geocentric or ECEF (Earth Centered Earth Fixed) Coordinates
  - Simulating Precipitation with SilverLining
  - Manipulating Time with SilverLining
  - Considerations for Multi-Channel, Multi-Viewport, and Multi-Context Setups
  - Managing Your Own Drawing
  - Crepuscular Rays
  - Advanced: using shadow maps
  - Using SilverLining with High Dynamic Range applications (HDR)
  - Creating environment maps from SilverLining
  - Selecting a Sky Model

  - Positioning Individual Clouds Within a Cloud Layer
  - Using SilverLining with Linear Color Space
  - Tweaking SilverLining's Appearance

- SilverLining Troubleshooting Tips Troubleshooting and performance tuning tips if you're getting unexpected results.

- SilverLining Support How to get help from Sundog Software, LLC when you're stuck.

- Copyright Notices Copyright notices

Chapter 2

SilverLining Licensing

In order to use SilverLining within your application, you must instantiate an Atmosphere object.

An Atmosphere must be constructed with a product name and license code. For example,

```
atm = new Atmosphere("Your Product Name", "Your License Code");
```

Invalid license names or codes will result an a dialog box being displayed when your application starts up, informing the user that this application is unlicensed. After five minutes of use, the application will terminate. This allows you to use the SilverLining SDK and evaluate it without paying for a license, but isn't appropriate for a finished application or serious development.

In order to use SilverLining without these restrictions, you must purchase a license code. Royalty-free licenses are available on a per-product basis. You'll get a license code which will fully unlock SilverLining for use in your application, access to SilverLining's complete source code, and access to top-tier developer support directly from the author.

Contact sales@sundog-soft.com or visit our website at http://www.sundog-soft.com/ for more details.

Chapter 3

SilverLining Support

You may contact us directly at support@sundog-soft.com.

Licensed developers will receive information on how to access top-tier developer support.

Chapter 4

Copyright Notices

The SilverLining Sky, 3D Cloud, and Weather SDK (tm) is Copyright (c) 2006-2015 Sundog Software, LLC.

All rights reserved worldwide.

Portions of the software are Copyright (c) 2012 - 2013, Lukas Hosek and Alexander Wilkie. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Please see the license.txt file included in the SDK for full license terms.

Chapter 5

SilverLining Troubleshooting Tips

5.1 "The application has failed to start because the application's configuration is incorrect."

If you receive this message when you run your application that links in SilverLining, it's due to a mismatch between the dynamic runtime libraries we linked SilverLining against, and the dynamic runtime libraries installed on your system. This is most commonly an issue under Visual Studio 2005, as a service update in July 2009 updated all of the runtime dependencies to a new version (see http://support.microsoft.com/kb/973544 .) This means that even if you have service pack 1 installed for Visual Studio 2005, you don't necessarily have the latest runtimes.

One solution is to ensure you have the most recent service packs installed for Visual Studio, and run Windows Update to apply any security patches that have been released since then. SilverLining is always linked against the latest updates from Microsoft. Be sure to install the latest redistributable dependencies from Microsoft with your application, to prevent this same issue on your user's systems.

An even better solution is to avoid the dynamic dependencies on the Visual Studio runtimes altogether. If you can, set your project settings to use the non-DLL versions of the runtimes. You can change this under the 'runtime library' property under the C++ / Language Settings property page. You may then link in the non-DLL versions of the SilverLining libraries as well, and avoid what's commonly known as "DLL hell" by statically linking the correct runtime directly into your application.

Another solution, if you have a full-source license for SilverLining, is to recompile the SilverLining libraries from source. That will cause them to link against whatever runtime libraries are installed on your system, which should also solve this issue. Again, if you're using the DLL runtimes, it's critical that you redistribute the same runtimes with your application.

5.2 SilverLining crashes or hangs at startup

Under Windows, there is a very good chance that you are linking against a different C++ runtime library than SilverLining was built against. Crashes inside an iterator or a mutex object inside of SilverLining are indications that this is what's going on. This sort of trouble can result when you are using the "ignore specific libraries" linker setting to work around linker errors from other third party libraries you are using.

Usually, this can worked around by ignoring the "msvcrt" variant that is causing trouble. Which one will depend on your specific configuration. For example, one customer who had trouble using the Multi-threaded debug (non-DLL) runtime was excluding libcmt.lib to work around other linker issues with the C runtime libraries, and got up and running by excluding msvcprtd.lib.

If you're using the evaluation version of the Windows SDK, SilverLining will display a dialog box when the Atmosphere constructor is invoked, warning you about the time restriction on the SDK. If you instantiate your Atmosphere object after your application is in full-screen mode, some applications will not be able to handle a dialog box being displayed while full-screen mode is active. To work around this, instantiate your Atmosphere object before entering full-screen mode.

If you are experiencing a crash the first time you call Atmosphere::DrawSky(), odds are you are calling DrawSky() from a different thread and/or rendering context than the one that was active when Atmosphere::Initialize() was called. Initialize(), DrawSky(), and DrawObjects() should all be called from the same context, and you should maintain individual Atmosphere objects per context.

Another situation may be OpenSceneGraph applications running with multiple monitors. The only OSG sample included with the SDK that handles multiple windows is the "OpenSceneGraph MultiWindow" example; please refer to it for guidance in using SilverLining in multi-window setups.

5.3 Atmosphere::Initialize() returns an error code

There are several cases that might cause the initialization of SilverLining to fail; here are the error codes returned from Atmosphere::Initialize for each:

E_RESOURCES_PATH_INVALID: This error will occur if the file SilverLining.config could not be opened within the resources path you specified in your call to Atmosphere::Initialize(). Usually, this means that your resource path is incorrect. Make sure the backslashes in your path are escaped, and that your path includes a trailing backslash, and double-check that the path you specified exists. The SilverLining SDK will be installed to different paths depending on whether you're on a 32-bit or 64-bit OS, for example.

E_CONFIGURATION_FILE_NOT_FOUND: If the file SilverLining.config within your resources directory could not be parsed for any reason, this error will be returned.

Generally, this means the config file has been modified incorrectly or is otherwise corrupt.

E_CANT_LOAD_RENDERER_DLL: SilverLining will construct a path to the appropriate renderer DLL based on whether you're using DirectX or OpenGL, VC8 or VC9, and 32 or 64 bit builds. These DLL's are stored within your resources directory; an example path might end up being resources/vc8/win32/SilverLiningOpenGLMT.dll for instance. If the renderer was unable to load this DLL at runtime, then this error will be returned. If you haven't moved these DLL's out of this location and you are specifying a valid value for the renderer parameter of Atmosphere::Initialize, please contact support@sundog-soft.com and indicate what renderer you're using, what version of Visual Studio, and whether you're using 32 or 64 bit versions of our libraries. Also send us a copy of your call to Atmosphere::Initialize().

E_CANT_INITIALIZE_RENDERER_SUBSYSTEM: This means SilverLining was unable to get some basic information from OpenGL or DirectX and was unable to set up its buffers for rendering to textures. Usually, this means you called Atmosphere::Initialize() before your rendering context was set up, or from outside of your rendering thread altogether. Be sure you're within the context you intend to use for rendering when calling Initialize(). SilverLining also requires a lockable backbuffer, if you're using an engine that does not provide this (such as Ogre3D,) set the renderoffscreen setting in resources/Silverlining.config to "yes" to disable direct use of the backbuffer.

E_CANT_LOAD_METABALL_TEXTURE: This error will return if the file wispymetaball.tga could not be loaded from within your specified resources directory path. Contact support@sundog-soft.com if you haven't moved or modified this file.

5.4 My sky is sideways / upside down / otherwise misoriented

Be sure to call Atmosphere::SetUpVector() and Atmosphere::SetRightVector() prior to calling Atmosphere::Initialize() so we know which way is up. Also, take care in setting the handedness parameter of Atmosphere::Initialize.

5.5 My sky doesn't appear or doesn't move with the camera

SilverLining assumes that the current modelview and projection matrices represent your camera's view when Atmosphere::DrawSky() is called. Some engines will reset their modelview and projection matrices outside of their own internal rendering code. When this happens, your sky will appear static or won't be in the right place at all. You may need to explicitly obtain your modelview and projection matrices, and set them directly via OpenGL or DirectX prior to calling Atmosphere::DrawSky().

5.6 SilverLining slows down my framerate too much

Having too many cumulus congestus clouds in a scene tends to be the main performance hog; the first thing you should do is ensure your cumulus congestus cloud layers aren't bigger than they need to be. The number of clouds we need to render increases with the square of the size of these layers. Reducing the density of these cloud layers will also help.

Beyond that, there are several settings in the file resources/SilverLining.config that you can adjust, in order to make your own trade-offs between performance and visual quality.

Start with the setting cumulus-congestus-voxel-dimension. The bigger you make this, the fewer voxels will be required to render each cumulus congestus cloud. Increasing this will help quite a bit, but the clouds will look a little less complex as a result.

If you are using stratocumulus clouds, their bottleneck is fragment processing - the resolution of the display will affect performance more than anything. You can tweak performance by adjusting the preprocessor definitions in the stratocumulus shaders themselves - open them up from the resources/shaders file for the renderer you are using, and you'll see ways of adjusting the sampling rates at the top of the shader.

If your viewpoint changes slowly, configuring SilverLining to use impostors more aggressively may help quite a bit. By default, the setting of cloud-backdrop-distance is set to a distance of 50km. Bringing this in closer to the camera will cause more cumulus clouds to be rendered to a ring of impostors surrounding the camera, which can yield big framerate improvements.

cull-interior-voxels will prevent SilverLining from rendering voxels that are unlikely to be visible, and helps a little bit as well.

Finally, if you notice a slowdown only when the sun is in view, this is probably due to the occlusion test being done to prevent the sun from producing lens flare when it's occluded by clouds or other objects. If you can live without that test, setting lens-flare-disable-occlusion = yes will help a lot (in Resources/SilverLining.config). If you're using OpenGL you can configure SilverLining to use asynchronous pixel buffer objects for this task, by setting lens-flare-use-pbo = yes. This will let you have occlusion tests on the sun for lens flare, but without blocking. This may cause issues if you are doing deferred rendering to a texture however - in that case, you're better off disabling lens flare occlusion entirely.

5.7 My framerate gets slower over time when I'm using SilverLining, for no apparent reason

This seems to be an issue specific to older NVidia drivers, related to our use of vertex buffer objects. To work around this, either update your drivers, or set the environment

variable SILVERLINING_NO_VBO to 1 on your system (and restart your development environment to pick it up.)

5.8 My sky or clouds are rendering funny

Try opening up the file resources/SilverLining.config in a text editor, and set "disableshaders" to "yes". Although we test SilverLining on a wide variety of systems, our shaders will sometimes compile in unexpected ways on new hardware we haven't encountered yet. Disabling shaders will cause SilverLining to fall back to more compatible methods of transforming its vertices, at a slight performance cost. If you run into this situation, please contact support@sundog-soft.com with your system information (especially what graphics card you're using) so we can support shaders on your system going forward.

5.9 I use an unusual coordinate system, and the clouds aren't showing up where I expect them

If you're using a geocentric coordiante system ("Earth Centered Earth Fixed"), be sure to read the section Using SilverLining in Geocentric or ECEF (Earth Centered Earth Fixed) Coordinates for important information on how to position cloud layers in these systems.

Some users render everything relative to the eyepoint. This causes problems, since SilverLining assumes that the modelview matrix you pass in represents the camera location and rotation, and that this matrix is consistent with the matrix OpenGL or DirectX is using for drawing. The best way around issues like this is to push your own special modelview matrix onto the stack using glPushMatrix prior to calling Atmosphere::DrawSky or Atmosphere::DrawObjects, then construct an equivalent modelview matrix that assumes the origin is at (0,0,0) instead of the eyepoint. Such a matrix will include the camera rotation and translation. Make this new matrix your modelview matrix, and pass it into Atmosphere::SetCameraMatrix - then call Atmosphere::DrawSky or Atmosphere::DrawObjects, and then restore your original modelview matrix prior to doing your own drawing.

5.10 SilverLining draws to all active render targets, but I only want it to draw to specific ones.

Atmosphere::DrawSky() and Atmosphere::DrawObjects() will render to any render targets currently active at the time. If you need to render only to specific render targets, you'll need to edit the fragment shaders inside the resources/shaders direc-

tory. OpenGL 2.0 shaders have a -frag.glsl extension, OpenGL 3.2+ shaders have a -frag.glsl15 extensions, and DirectX shaders have a .fx extension. Under OpenGL, you'll want to replace our assignments to gl_FragColor with specific elements of gl_FragData[] instead.

5.11 My problem isn't listed here!

Send us a note at support@sundog-soft.com. Even if you haven't purchased a license yet, we'll do what we can to help get you up and running.

Chapter 6

Overview of SilverLining

6.1 SDK Contents

The SilverLining SDK includes the following subdirectories:

- **lib:** The SilverLining libraries built against various runtime libraries. You'll need to link one of these into your project, as described in the table below.
- **include:** The public header files for use with the SilverLining libraries.
- **samplecode:** Sample code for simple OpenGL and Direct3D applications that use SilverLining. You'll also find a modified "osgviewer" application that integrates SilverLining under OpenSceneGraph, and a modified Ogre3D demo application.
- **resources:** Graphics, shaders, and other resources needed by SilverLining, including the DLL's specific to OpenGL, OpenGL 3.2 core, and DirectX9, 10, 11, or 11.1 for Windows users. These files must be redistributed with your application.
- **docs:** Documentation for SilverLining (you're reading it!)

The **lib** directory contains several versions of the SilverLining library for linking against projects that use different runtime libraries. Builds are available for Microsoft Visual C++ 6, 2003, 2005, 2008, 2010, 2012, 2013, and 2015 in the lib/vc6, lib/vc7, lib/vc8, lib/vc9, lib/vc10, lib/vc11, lib/vc12, and lib/vc14 directories, respectively. Within the vc8, 9, 10, 11, 12 and 14 directories you'll find x64 and win32 subdirectories that contain 64-bit and 32-bit builds, respectively. Linux developers will find the static library linux/libSilverLining.a in here, after building the SDK with "make."

6.1.1 Windows Build Settings

The Windows SDK installer will set up the environment variable SILVERLINING_-PATH, which you can use to safely specify absolute paths to the SilverLining libraries and headers.

To check what runtime library you are using in Visual Studio 2005, open up the project properties, and view the C/C++ "Code Generation" settings. Add the library according to the chart below, by adding it to the "Additional Dependencies" in your Linker / Input settings. Finally, be sure that your linker path is set to include the appropriate lib directory for your development environment (for example, lib/vc8/win32 - see "additional library directories" under the Linker / General tab.)

Also, you will need to link in the system library winmm.lib in order to obtain the definition of timeGetTime().

DirectX11 users need to link in the build of D3DX11Effects.lib specific to your runtime and compiler under the third-party/Effects11 directory. This is not needed when using the DirectX11.1 renderer, as our DirectX 11.1 support has no dependencies on the effects library, D3DX interfaces, or the D3Dcompiler DLL.

DirectX 9-11 users should be building with the June 2010 version of the DirectX SDK. DirectX 11.1 users may build against the DirectX support included in the Windows 8.1+ SDK.

Project files and libraries are included for Visual Studio 6 (VC6), Visual Studio .NET 2003 (VC7), Visual Studio 2005 (VC8), Visual Studio 2008 (VC9), Visual Studio 2010 (VC10), Visual Studio 2012 (VC11), Visual Studio 2013 (VC12), and Visual Studio 2105 (VC14). Our Visual Studio 2008 and 2010 libraries are linked against version 7 of the Microsoft SDK; Visual Studio 2005 is linked against version 6.0a. Visual Studio 2012 is linked against version 8; at present, we only support desktop applications under Windows 8 and not Metro apps. Our libraries are built with the latest service packs installed for each version of Visual Studio; you may encounter runtime linking errors if your Visual Studio is not up to date.

Remember - Visual Studio version numbers and the years don't necessarily match up! For example, Visual Studio 2008 is VC9, not VC8, and Visual Studio 2012 is VC11, not VC12. Make sure you are linking against the correct libraries for the compiler you are using.

Support for VC6 and VC7 is limited. Library builds are not provided for linking against custom renderers with the VC6 and VC7 libraries, and only the OpenGL and DirectX9 renderers will work with them.

Runtime Library	**SilverLining Library**
Multi-threaded	SilverLining-MT.lib
Multi-threaded Debug	SilverLining-MTD.lib
Multi-threaded DLL	SilverLining-MT-DLL.lib
Multi-threaded Debug DLL	SilverLining-MTD-DLL.lib

The libraries are built with default byte alignment. If you have specific byte alignment requirements for your application, contact support@sundog-soft.com for any specialized builds you may require.

If you encounter an exception or crash when calling a SilverLining method, odds are you are linking in a C standard runtime library that is different from the one SilverLining was bulit against. This is especially likely if you are ignoring specific runtime libraries in your linker settings, to work around other linker issues with other third party libraries. Usually this is cleared up by also ignoring the msvcrt library variant that's causing the trouble. Don't hesitate to contact support@sundog-soft.com if you run into this sort of trouble. Crashes inside an iterator or mutex object within SilverLining are symptoms of this sort of issue.

The libraries listed above will dynamically link in a DirectX9, DirectX10, DirectX11, or OpenGL rendering library at runtime. Some developers prefer to avoid runtime DLL dependencies; and for customers porting SilverLining to game consoles, this may be required. Licensed users with the full source will find build configurations for the SilverLining library in its project file for statically linking in these renderers, avoiding the runtime dependency on our renderer DLL's.

6.1.2 Linux Build Settings and Dependencies

To successfully link against SilverLining for Linux, you must have OpenGL installed on your system. Make sure you have the latest drivers for your graphics card installed.

After building the SDK using "make", SilverLining's libraries will be in the lib/linux subdirectory. The OpenGL sample code provided includes a makefile that illustrates how to link in SilverLining. You'll find three libraries here:

- **libSilverLiningOpenGL.a** - SilverLining for any OpenGL context from version 1.4 up; also compatible with 3.2 and above contexts with backward compatibility enabled.
- **libSilverLiningOpenGL32Core.a** - SilverLining compatible with OpenGL 3.2 core contexts and above.
- **libSilverLining-norenderer.a** - SilverLining with no rendering calls linked in at all. When using this library, you must implement your own renderer that implements the interface defined in SilverLiningDLLCommon.h.

In short, add the lib directory of the sdk to your library path (-L), add the include directory to your include path (-I), and include the libraries SilverLining, GL, and pthreads (-l).

That's all there is to it. Licensed users with full source code receive unobfuscated source code to build from.

6.1.3 Building with C#

You'll find an example of C# integration with SilverLining in the C# Example folder in the sample applications. It's composed of four different projects, which you can build using Visual Studio 2010 Professional. Pre-compiled DLL's of the C# interface are also provided for you, if you don't have the Pro edition.

- **SilverLiningInt** - This is a DLL wrapper for the SilverLining native libraries, which may be used to integrate with other interpreted languages as well.
- **SilverLiningCLR** - This is a C# wrapper over the native DLL, providing C# access to our unmanaged code. You'll find wrappers for the Atmosphere, CloudLayer, and AtmosphericConditions classes, enabling you to do almost everything in C# that you can do using the C++ public interface. Refer to the C++ class documentation for more details on the methods exposed.
- **OpenGLExample** - This example C# application uses SilverLiningCLR to create a scene containing a dynamic skybox and cumulus clouds. You'll need to install OpenTK (available at www.opentk.com) to build this sample.
- **XNAExample** - This example is based on a standard XNA Game Framework 4.0 project, and illustrates some subtleties surrounding integration of SilverLining with XNA and DirectX 9. Notably, obtaining the underlying IDirect3DDevice9 pointer to pass into Atmosphere::Initialize(), and forcing XNA into a BeginScene()/EndScene() drawing block prior to calling Atmosphere::DrawSky().

If you're using Managed DirectX, the only trick is to pass the UnmanagedComPointer member of the Device object as the last parameter in AtmosphereCLR::Initialize().

When redistributing a C# application using SilverLining, ensure that the SilverLiningInt.dll is in the application's path, and that the correct path to the resources folder is passed into Atmosphere::Initialize().

6.2 Distributing SilverLining with Your Application

You must ship the **resources** subdirectory, in its entirety, with your application. You will specify where this subdirectory is installed when you call Atmosphere::Initialize().

If your project is linked against the dynamically linked runtime libraries, be sure that the required Microsoft DLL's are distributed with your application as well.

DirectX users must install the latest DirectX runtime libraries (June 2010 or newer.) DirectX11 users must furthermore install the Direct3D11 dll's on systems that do not have them using the D3D11InstallHelper or an equivalent scheme. See http://msdn.microsoft.com/en-us/
and http://msdn.microsoft.com/en-us/library/ee416644(VS.85).aspx
for more information.

If you are developing with Visual Studio 2012 and the Windows 8 SDK, and using DirectX, you also need to redistribute the runtime shader compilation DLL's alongside your application's executable. You'll find these inside the Windows 8 SDK under %ProgramFiles(x86)%\Windows Kits\8.0\Redist\D3D\<arch> where <arch> is x86 and x64. You must also install the runtimes for the June 2010 DirectX DLL to ensure the D3DX methods that are not included in the Windows 8 SDK are available.

Linux application developers must ensure that OpenGL is installed on the target systems, as well as the latest graphics drivers for their 3D video card.

If you are distributing a commercial application with SilverLining, please remember to include the required copyright notices for SilverLining in your application. See your license agreement for more details. We'd also appreciate including our name and/or logo along with your other middleware credits.

6.3 Overview of SilverLining

Using SilverLining isn't complicated, and involves understanding only a few classes and how they interact with each other. While developing an application with SilverLining, you'll likely encounter these classes in this order:

First, you'll create an SilverLining::Atmosphere object, which is the primary interface to SilverLining for your application. After instantiating it, it needs to be initialized so that it knows if you're using OpenGL, OpenGL 3.2 Core, DirectX 9, DirectX 10, or DirectX 11.

Next, you'll want to create a SilverLining::AtmosphericConditions object, in order to configure your Atmosphere to your liking.

The first things you'll want to configure in AtmosphericConditions are the time and location you wish to simulate. To do this, you'll need to instantiate and configure a LocalTime and a Location class, and pass these into SilverLining::AtmosphericConditions::SetTime() and SilverLining::AtmosphericConditions::SetLocation().

If you want to simulate wind in your scene to make your clouds move, just set up a SilverLining::WindVolume class and pass that into SilverLining::AtmosphericConditions::SetWind().

If you want to add clouds to your scene, the easiest way to get started is to use SilverLining::AtmosphericConditions::SetPresetConditions(). This will let you quickly set up fair, partly cloudy, mostly cloudy, or overcast conditions with one line of code.

For more control of the cloud conditions, you'll need to create some SilverLining::CloudLayer objects and pass them into SilverLining::AtmosphericConditions::AddCloudLayer(). CloudLayer objects are created with the SilverLining::CloudLayerFactory class, which allows you to create cumulus congestus, cumulus mediocris, cumulonimbus, stratus, nimbostratus, or cirrus cloud decks. After creating the CloudLayer you want, configure it for the desired dimensions, precipitation, and cloud coverage, call SilverLining::CloudLayer::SeedClouds() to populate the cloud layer with clouds, and then pass

it into your AtmosphericConditions.

Once your AtmosphericConditions are set up to your liking, pass them into your Atmosphere using SilverLining::Atmosphere::SetConditions().

That's all the setup that's required. Now, to actually see the simulated sky and clouds in your scene, only three more things are necessary: at the beginning of each frame, pass your modelview and projection matrices into SilverLining using SilverLining::Atmosphere::SetCameraMatrix() and SilverLining::Atmosphere::SetProjectionMatrix(). Then, call SilverLining::Atmosphere::DrawSky() before rendering the objects in your scene, and call SilverLining::Atmosphere::DrawObjects() at the end of your frame.

That's really all there is to it; we've discussed every class you need to know about already. Users who want to take advantage of the lighting and fog simulation provided by SilverLining can obtain this information from the Atmosphere class. Details on that may be found in Using SilverLining.

Chapter 7

Using SilverLining

7.1 A simple example

7.1.1 Initialization

Complete sample code for both OpenGL and Direct3D is found in the SampleCode/OpenGLExample, SampleCode/DirectX9Example, SampleCode/DirectX10Example, and SampleCode/DirectX11Example directories of the SDK (The Linux SDK only includes OpenGL). Here's an overview of some of the key bits of code found in each.

When your application starts up, you'll want to initialize OpenGL or Direct3D, and create and initialize an Atmosphere object. If you want your clouds to be different every time your application is run, be sure to seed the random number generator as well.

Under DirectX9, it's recommended that you create your IDirect3DDevice object without the D3DCREATE_PUREDEVICE flag. Silverlining needs to call GetTransform and GetRenderState on your device, in order to put things back the way we found them when we're done drawing. We also recommend creating your device with the D3DPRESENTFLAG_LOCKABLE_BACKBUFFER flag, which will allow you more flexibility in configuring how SilverLining does its lighting (specifically, this will allow you to set render-offscreen to "no" in SilverLining.config.)

If your Atmosphere initializes successfully, you can go ahead and start configuring it to add your clouds, and specify the time and location you wish to simulate. We'll get into that next. If it does not initialize successfully, it's likely that the user needs to update the graphics driver. SilverLining has fallback cases for most cases where compatibility might be an issue, but some older drivers have issues even with standard functionality.

```
// If you want different clouds to be generated every time, remember to seed the
// random number generator.
srand(time(NULL));
```

```
// Initialize your Direct3D device object, glut, or wgl as appropriate
InitializeGraphicsSubsystem();

// Instantiate an Atmosphere object. Substitute your own purchased license name a
      nd code here.
atm = new Atmosphere("Your Company Name", "Your License Code");

// Tell SilverLining we're rendering in OpenGL, the Resources directory is 2 dire
      ctories
// above the working directory, and we're using a right-handed coordinate system.

if (atm->Initialize(Atmosphere::OPENGL, "..\\..\\Resources\\", true, 0) == Atmosp
      here::E_NOERROR)
{
    // Tell SilverLining what your axis conventions are.
    atm->SetUpVector(0.0, 1.0, 0.0);
    atm->SetRightVector(1.0, 0.0, 0.0);


    // Set up all the clouds
    SetupAtmosphericConditions();

    // Configure where and when we want to be
    SetTimeAndLocation();

    // Start rendering.
    glutMainLoop();
}
```

Let's flesh out SetupAtmosphericConditions() from above. Once you have an Atmosphere object initialized, you can access its AtmosphericConditions object to do things like add cloud decks, change the time of day, and simulate wind. To get up and running fast, you can set up cloud conditions with a single line of code like this:

```
    atm->GetConditions()->SetPresetConditions(AtmosphericConditions::MOSTLY_CLOUD
      Y, *atm);
```

But, you might want finer control of the cloud layers. Let's write a code snippet to add a cirrus deck, a cumulus congestus deck, and make it a windy day. The code is self-explanatory:

```
static void SetupCirrusClouds()
{
    CloudLayer *cirrusCloudLayer;

    cirrusCloudLayer = CloudLayerFactory::Create(CIRRUS_FIBRATUS);
    cirrusCloudLayer->SetBaseAltitude(8000);
    cirrusCloudLayer->SetThickness(500);
    cirrusCloudLayer->SetBaseLength(100000);
    cirrusCloudLayer->SetBaseWidth(100000);
    cirrusCloudLayer->SetLayerPosition(0, 0);
    cirrusCloudLayer->SeedClouds(*atm);
```

```
    atm->GetConditions()->AddCloudLayer(cirrusCloudLayer);
}
```

The above routine will create a Cirrus cloud (the high, wispy ones) at an altitude of 8,000 meters and 100 km across. It's centered at the camera position.

Anytime you create a new CloudLayer, you must first

- Instantiate it using the CloudLayerFactory class factory.
- Set its size, coverage, and position parameters
- Call CloudLayer::SeedClouds() to populate the layer with clouds
- Add the layer to your atmosphere via AtmosphericConditions::AddCloudLayer()

Similarly, let's set up a cumulus congestus deck:

```
// Add a cumulus congestus deck with 40% sky coverage, which stays centered aroun
      d the camera position.
static void SetupCumulusCongestusClouds()
{
    CloudLayer *cumulusCongestusLayer;

    cumulusCongestusLayer = CloudLayerFactory::Create(CUMULUS_CONGESTUS);
    cumulusCongestusLayer->SetIsInfinite(true);
    cumulusCongestusLayer->SetBaseAltitude(1500);
    cumulusCongestusLayer->SetThickness(100);
    cumulusCongestusLayer->SetBaseLength(30000);
    cumulusCongestusLayer->SetBaseWidth(30000);
    cumulusCongestusLayer->SetDensity(0.4);
    cumulusCongestusLayer->SetLayerPosition(0, 0);
    cumulusCongestusLayer->SetFadeTowardEdges(true);
    cumulusCongestusLayer->SetAlpha(0.8);
    // Enable convection effects, but not growth:
    cumulusCongestusLayer->SetCloudAnimationEffects(0.1, false, 0);
    cumulusCongestusLayer->SeedClouds(*atm);

    atm->GetConditions()->AddCloudLayer(cumulusCongestusLayer);
}
```

Our SetupAtmosphericConditions function will set up the simulated wind, call the above two functions to create the cirrus and cumulus cloud decks, and finally set the simulated visibility - which will affect the fog effects on the clouds themselves:

```
// Configure SilverLining for the desired wind, clouds, and visibility.
static void SetupAtmosphericConditions()
{
    // Set up wind blowing south at 50 meters/sec
    WindVolume wv;
    wv.SetDirection(180);
    wv.SetMinAltitude(0);
```

```
    wv.SetMaxAltitude(10000);
    wv.SetWindSpeed(50);
    atm->GetConditions()->SetWind(wv);

    // Set up the desired cloud types.
    SetupCirrusClouds();
    SetupCumulusCongestusClouds();

    // Set visibility in meters
    atm->GetConditions()->SetVisibility(100000);
}
```

If you wish to simulate a particular place and time, you should also set that up in your initialization. You can also change this at any time while the application is running. Be sure that the time zone you specify in the LocalTime object is consistent with the longitude you specify in the Location object, or else you'll be very confused by the results! (It's also OK to specify all times as in the GMT time zone if you want to use UTC time consistently instead of local times.)

```
// Sets the simulated location and local time.
// Note, it's important that your longitude in the Location agrees with
// the time zone in the LocalTime.
static void SetTimeAndLocation()
{
    Location loc;
    loc.SetLatitude(45);
    loc.SetLongitude(-122);

    LocalTime tm;
    tm.SetYear(1971);
    tm.SetMonth(8);
    tm.SetDay(5);
    tm.SetHour(14);
    tm.SetMinutes(0);
    tm.SetSeconds(0);
    tm.SetObservingDaylightSavingsTime(true);
    tm.SetTimeZone(PST);

    atm->GetConditions()->SetTime(tm);
    atm->GetConditions()->SetLocation(loc);
}
```

7.1.2 Infinite Cloud Layers

Cumulus congestus, cumulus mediocris, stratus, cirrus, stratocumulus, and cirrocumulus cloud layers may be modified by using the CloudLayer::SetIsInfinite() method.

Infinite cloud layers will stay centered at the camera location; as the camera moves, clouds that leave the bounding area defined by the cloud layer's length and width will be repositioned to pop in where the camera's moving toward. Similarly, if wind blows clouds outside of the cloud layer, they will wrap around to the other side of the layer.

Infinite cloud layers allow you to not worry about positioning the cloud layer, or setting up multiple cloud layers to cover large areas.

This gives you the effect of an "infinite" cloud layer, where the clouds will never blow away and you can't move the camera away from them. The larger the length and width you create the cloud layer with, the less noticable the popping will be as clouds are repositioned - especially if the clouds are being fogged in the distance by setting AtmosphericConditions::SetVisibility() to a value similar to the cloud layer's dimensions. You can also use CloudLayer::SetFadeTowardEdges() to fade the clouds out before they reach the edge of the layer, ensuring popping is never visible.

CloudLayer::SetIsInfinite() should be called when initializing the cloud layer; by default, cloud layers are not infinite. Cumulonimbus cloud layers are not affected by SetIsInfinite(), as they only contain a single cloud.

7.1.3 Cloud Animation Effects

Cumulus cloud layers support optional animation effects, to simulate convection and growth of the clouds at runtime. Use the CloudLayer::SetCloudAnimationEffects() method to control these effects.

The first parameter controls whether individual cloud puffs will rotate at a random rate, giving the clouds a convection effect. The value is the maximum rotation in radians per second. There is no performance cost to this effect.

The second parameter controls a real dynamic cloud growth simulation, powered by cellular automata. This does incur a performance hit on the CPU, but in most cases it is negligable. If you do enable growth effects, the third parameter lets you control how "evolved" the cloud is at startup time. Leave it set to zero to start off with fully grown clouds that change their shape gradually, or to a smaller value such as one to watch the clouds form in real-time. The fourth parameter controls the number of seconds between iteration of the cellular automata. Set this to longer values for slower growth effects, or pass 0 to use the default values.

When used together with time-lapse effects (by using a custom MillisecondTimer class,) dramatic effects are possible with clouds shifting and growing over time.

7.1.4 Integrating SilverLining with your Rendering Loop

That's pretty much it for initialization. Now, how do you integrate SilverLining with the rendering of each frame? It's quite simple. When you render a frame of animation, you'll want to follow these steps:

- Set the modelview and projection matrices to represent your camera position and scene.
- Call Atmosphere::DrawSky() to draw the sky and compute lighting information.

- Set up your scene's lighting by using the information returned from Atmosphere::GetSunOrMoonPosition(), Atmosphere::GetSunOrMoonColor(), and Atmosphere::GetAmbientColor().
- Set up your scene's fog by using the information returned from Atmosphere::GetFogEnabled() and Atmosphere::GetFogSettings().
- Draw your scene's objects.
- Call Atmosphere::DrawObjects() to draw the clouds.

Here's an example of the render loop under OpenGL:

```
void Display()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(45.0, aspectRatio, 2, 100000);

        // Increment the yaw each frame to spin the camera around
        yaw += 0.05;

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glRotatef(-10, 1, 0, 0);
        glRotatef(yaw, 0, 1, 0);
        glTranslatef(0, -100, 0);

        // Pass in the view and projection matrices to SilverLining.
        double mv[16], proj[16];
        glGetDoublev(GL_MODELVIEW_MATRIX, mv);
        glGetDoublev(GL_PROJECTION_MATRIX, proj);
        atm->SetCameraMatrix(mv);
        atm->SetProjectionMatrix(proj);

        // After setting up your projection and modelview matrices to reflect the
       current
        // camera position, call Atmosphere::DrawSky() to draw the sky and do the
       lighting
        // pass on the clouds, if necessary.
        atm->DrawSky(true);

        // Now, do all your own drawing...
        SetSceneLighting();
        SetSceneFog();
        DrawGroundPlane();

        // When you're done, call Atmosphere::DrawObjects() to draw all the cloud
      s from back to front.
        atm->DrawObjects();

        // Now swap the back and front buffers.
        glutSwapBuffers();

        glutPostRedisplay();
}
```

And a main rendering loop under DirectX 9... note that this example uses a right-handed coordinate system. The handedness must match what you specified when initializing the Atmosphere object. This example doesn't handle lost devices; see the Atmosphere::D3D9DeviceLost() and Atmosphere::D3D9DeviceReset() methods for proper handling of that.

DirectX10 and DirectX11 would be similar, but instead of setting the view and projection matrices into the device's fixed function pipeline as you would in DirectX9, you'll need to pass those to your vertex programs directly. See the DirectX 10 and 11 sample code provided with the SDK for a complete example.

```
static void RenderFrame(HWND hWnd)
{
        static float lastTime = (float)timeGetTime();

        if (atm && device)
        {
                D3DXMATRIX Rot, Yaw, Pitch;
                D3DXMatrixRotationX(&Pitch, -10.0f * (3.14f / 360.0f));
                D3DXMatrixRotationY(&Yaw, yaw);
                D3DXMatrixMultiply(&Rot, &Yaw, &Pitch);
                D3DXMATRIX Pos;
                D3DXMatrixTranslation(&Pos, 0, -100, 0);
                D3DXMATRIX view;
                D3DXMatrixMultiply(&view, &Pos, &Rot);

                device->SetTransform(D3DTS_VIEW, &view);

                //
                // Set projection matrix.
                //
                D3DVIEWPORT9 vp;
                device->GetViewport(&vp);

                D3DXMATRIX proj;
                D3DXMatrixPerspectiveFovRH(
                        &proj,
                        45.0 * (D3DX_PI / 180.0),
                        (float)vp.Width / (float)vp.Height,
                        2.0f,
                        200000.0f);
                device->SetTransform(D3DTS_PROJECTION, &proj);

                // Set view and proj matrices with SilverLining
                if (atm)
                {
                        double pView[16], pProj[16];
                        int i = 0;
                        for (int row = 0; row < 4; row++)
                        {
                                for (int col = 0; col < 4; col++)
                                {
                                        pView[i] = view(row, col);
                                        pProj[i] = proj(row, col);
                                        i++;
                                }
```

```
                        }

            atm->SetCameraMatrix(pView);
            atm->SetProjectionMatrix(pProj);
        }

                device->BeginScene();

                // Call DrawSky after scene has begun and modelview / projection
      matrices
                // properly set for the camera position. This will draw the sky i
      f you pass true.
                atm->DrawSky(true);

                // Now, do all your own drawing...
                SetSceneLighting();
                SetSceneFog();
                DrawGroundPlane();

                // Call DrawObjects to draw all the clouds from back to front.
                atm->DrawObjects();

                device->EndScene();

                device->Present(0, 0, 0, 0);

                // Trigger another redraw.
                InvalidateRect(hWnd, NULL, FALSE);

                lastTime = currTime;
        }
}
```

7.1.5 Lighting Your Scene with SilverLining

In order to light the objects in your scene consistently with the appearance of the sky, SilverLining allows you to query for its modeled directional and ambient light information. It's easy to use this information to light your scene. Here's an example of setting up lighting under OpenGL using SilverLining as guidance:

```
void SetSceneLighting()
{
        float x, y, z, r, g, b, ra, ga, ba;
        atm->GetSunOrMoonPosition(&x, &y, &z);
        atm->GetSunOrMoonColor(&r, &g, &b);
        atm->GetAmbientColor(&ra, &ga, &ba);

        GLfloat light_ambient[] = {ra, ga, ba, 1.0};
        GLfloat light_diffuse[] = {r, g, b, 1.0};
        GLfloat light_specular[] = {0.0, 0.0, 0.0, 1.0};
        GLfloat light_position[] = {x, y, z, 0};

        glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
```

```
        glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glEnable(GL_LIGHT0);

        GLfloat mat_amb_diff[] = {1.0, 1.0, 1.0, 1.0};
        GLfloat no_mat[] = {0, 0, 0, 0};
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_amb_diff);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_amb_diff);
        glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
        glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);
}
```

And the same function for setting the scene's lighting, under DirectX9:

```
static void SetSceneLighting()
{
        D3DXCOLOR light_ambient, light_diffuse, light_specular;
        D3DXVECTOR3 light_position;

        atm->GetSunOrMoonPosition(&light_position.x, &light_position.y, &light_po
      sition.z);
        atm->GetSunOrMoonColor(&light_diffuse.r, &light_diffuse.g, &light_diffuse
      .b);
        atm->GetAmbientColor(&light_ambient.r, &light_ambient.g, &light_ambient.b
      );

        light_diffuse.a = light_ambient.a = 1.0;

        D3DLIGHT9 light;
        ::ZeroMemory(&light, sizeof(light));

        light.Type = D3DLIGHT_DIRECTIONAL;
        light.Ambient = light_ambient;
        light.Diffuse = light_diffuse;
        light.Specular = D3DXCOLOR(1, 1, 1, 1);
        light.Direction = D3DXVECTOR3(-light_position.x, -light_position.y, -ligh
      t_position.z);

        if (device)
        {
                device->SetLight(0, &light);
                device->LightEnable(0, true);
                device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
                device->SetRenderState(D3DRS_SPECULARENABLE, false);
        }
}
```

DirectX 10 and 11 would be similar, but you'd pass the lighting values and position into your shaders instead of to the device.

7.1.6 Fog Effects with SilverLining

Finally, SilverLining will also give you guidance on how to configure fog for your scene. If you're inside a stratus cloud deck or inside precipitation, SilverLining will

request that you set the fog in your scene appropriately to simulate being inside a thick cloud or inside rain or snow. If Atmosphere::GetFogEnabled() returns true, then SilverLining is asking that you query Atmosphere::GetFogSettings() to obtain information about the fog volume you're currently inside.

Even if you're not inside or under a cloud, SilverLining can help you set your fog to blend your distant terrain into the sky. SilverLining constantly computes the average color of the sky at the horizon within the current field of view, and makes this accessible via Atmosphere::GetHorizonColor(). Setting the fog color to this, and setting the density consistently with the visibility you passed earlier to Atmosphere::SetVisibility(), will yield realistic results for scenes with light haze.

It is possible to simulate a thick layer of colored haze hugging the ground. This can be useful if Atmosphere::GetHorizonColor() does not produce desirable results for your application. Effectively, this allows you to blend the skybox to a specified color as it approaches the horizon. If you fog your terrain with this same color, you can obscure the horizon line quite nicely for applications that do not render terrain all the way out to the horizon.

The haze layer is set via Atmosphere::SetHaze(). You may simulate any depth of haze, any color, and any density. Set the depth to 0 to disable haze. It's important to realize that no lighting is performed on the haze; if you want the sky to blend toward a darker color at night, you must pre-multiply the haze color by the light in the scene.

AtmosphericConditions::SetVisibility() may be used to apply atmospheric perspective effects to the clouds in the scene - it causes the clouds to blend into the sky with distance. It does not fog the sky itself - for thicker fog, use Atmosphere::SetHaze() for volumetric-style fog, or AtmosphericConditions::SetFog() for exponential fog. Think of SetVisibility() of just simulating particulate matter in the atmosphere that affects atmospheric perspective; it only makes sense with relatively high visibilities. If you're really simulating being in fog, use SetFog() instead.

Here's an example of setting fog under OpenGL, using the GetHorizonColor() method instead of a haze layer of a specified color. Note that this code will not result in the sky being fogged; it only provides guidance to your application for fogging the objects in your scene toward a color that blends with the horizon. For thicker fog that does obscure the sky, you'll want to use Atmosphere::SetHaze(), or AtmosphericConditions::SetFog() with a backbuffer cleared to your fog color, in addition to this code.

```
void SetSceneFog()
{
        glEnable(GL_FOG);
        glFogi(GL_FOG_MODE, GL_EXP);

        float hazeDensity = 1.0 / kVisibility;

        // Decrease fog density with altitude, to avoid fog effects through the v
      acuum of space.
        static const double H = 8435.0; // Pressure scale height of Earth's atmos
      phere
        double isothermalEffect = exp(-(atm->GetConditions()->GetLocation().GetAl
```

```
titude() / H));
  if (isothermalEffect <= 0) isothermalEffect = 1E-9;
  if (isothermalEffect > 1.0) isothermalEffect = 1.0;
  hazeDensity *= isothermalEffect;

  bool silverLiningHandledTheFog = false;

  if (atm->GetFogEnabled())
  {
          float density, r, g, b;
          // Note, the fog color returned is already lit
          atm->GetFogSettings(&density, &r, &g, &b);

          if (density > hazeDensity)
          {
                  glFogf(GL_FOG_DENSITY, density);

                  GLfloat fogColor[4] = {r, g, b, 1.0};
                  glFogfv(GL_FOG_COLOR, fogColor);

                  silverLiningHandledTheFog = true;
          }
  }

  if (!silverLiningHandledTheFog)
  {
          GLfloat fogColor[4];
          atm->GetHorizonColor(yaw, 0, &fogColor[0], &fogColor[1], &fogColo
r[2]);
          glFogfv(GL_FOG_COLOR, fogColor);

          glFogf(GL_FOG_DENSITY, hazeDensity);
  }
}
```

And here's the equivalent DirectX9 code for setting fog:

```
static void SetSceneFog()
{
  DWORD fogColor;
  float density, r, g, b;

  // If you're inside a cloud, SilverLining will request that you set the f
og accordingly.
  if (atm->GetFogEnabled())
  {
          atm->GetFogSettings(&density, &r, &g, &b); // This fog color is p
re-lit
  }
  else // Otherwise, setting the fog to the average color of the sky at the
 horizon works well.
  {
          atm->GetHorizonColor(yaw, &r, &g, &b);
          density = 1.0f / kVisibility;

  // Decrease fog density with altitude, to avoid fog effects through the v
```

```
      acuum of space.
        static const double H = 8435.0; // Pressure scale height of Earth's atmos
      phere
        double isothermalEffect = exp(-(atm->GetConditions()->GetLocation().GetAl
      titude() / H));
        if (isothermalEffect <= 0) isothermalEffect = 1E-9;
        if (isothermalEffect > 1.0) isothermalEffect = 1.0;
        density *= isothermalEffect;
        }

        BYTE cr, cg, cb, ca;
        cr = (BYTE)(r * 255.0);
        cg = (BYTE)(g * 255.0);
        cb = (BYTE)(b * 255.0);
        ca = 255;

        fogColor = D3DCOLOR_RGBA(cr, cg, cb, ca);

        // Enable fog blending.
        device->SetRenderState(D3DRS_FOGENABLE, TRUE);

        // Set the fog color.
        device->SetRenderState(D3DRS_FOGCOLOR, fogColor);

        // Set fog parameters.
        device->SetRenderState(D3DRS_FOGTABLEMODE, D3DFOG_EXP);
        device->SetRenderState(D3DRS_FOGDENSITY, *(DWORD *)(&density));
}
```

Again, DirectX10 and 11 is similar, but you would need to pass in the fog parameters to your shaders instead of to the device.

That's all there is to it! Using the techniques above, you'll be able to integrate SilverLining's sky and cloud rendering into your 3D application, and light and fog the objects in your scene consistently with the sky and clouds rendered by SilverLining.

If your application specifies its own fog - for example, you're simulating being inside a fog bank - you can tell SilverLining to fog its clouds to a specified color and density. See the AtmosphericConditions::SetFog() method for more information. When you're in thick fog, you'll usually want to just clear your back buffer to the fog color and not call SilverLining at all. The sky won't be visible in such conditions anyhow. Remember to call ClearFog() when you leave the simulated fog bank, as calling SetFog() will override all other atmospheric perspective effects in SilverLining.

7.2 Drawing Your Skybox Last Instead of First

Our instructions above describe calling Atmosphere::DrawSky() as the first thing in your frame in order to keep things simple and as compatible as possible with other engines. As you're getting your SilverLining integration up and running, it makes sense to start with this approach. However, in some situations, drawing the sky box last instead of first will actually lead to performance benefits.

The issue is that if you draw your sky box first, a lot of the box ends up getting overdrawn by the objects in your scene, which is a waste of fill rate. By drawing the sky box last, with depth reads enabled, only parts of the scene with a still-cleared depth buffer will be filled with the sky.

To do this, you must clear the depth buffer yourself at the beginning of each frame (but don't clear the color buffer.) Draw the non-translucent objects in your scene with depth writes on. Then, call Atmosphere::DrawSky() with the clearDepth parameter set to false - this will tell SilverLining that you don't want DrawSky() to clear the depth buffer before drawing the sky box. Finally, call Atmosphere::DrawObjects() to draw the clouds and precipitation, and the translucent objects in your scene.

Some engines don't use the depth buffer at all, and use depth textures instead - this technique will not work in such situations. However, for engines that do rely on the depth buffer, this technique can lead to slight gains in performance if your application is fill-rate bound.

7.3 Integration with Multi-Threaded Renderers

Some engines may perform updates, culling, and drawing in separate passes or threads. SilverLining allows you to perform these operations separately for better performance in such multi-threaded engines (such as OpenSceneGraph.)

By default, Atmosphere::DrawSky() will perform updates, culling, and drawing of the sky. However, you may call Atmosphere::UpdateSkyAndClouds() prior to Atmosphere::DrawSky() to update the sky's ephemeris model and the shapes and positions of the clouds at a more opportune time. If Atmosphere::DrawSky() is called without a call to Atmosphere::UpdateSkyAndClouds() before the prior call to Atmosphere::DrawSky(), DrawSky() will call UpdateSkyAndClouds() on its own prior to rendering.

Similarly, Atmosphere::CullObjects() may be used to cull the clouds outside of the call to Atmosphere::DrawObjects(). If CullObjects() is not called prior to DrawSky(), DrawSky() will call CullObjects() itself.

Once Atmosphere::UpdateSkyAndClouds() has bene called, you may query the bounding volume of the clouds that are in the scene via Atmosphere::GetCloudBounds().

Thread-safety is in place to prevent updates, culling, and drawing from happening simultaneously within SilverLining.

7.4 Memory Management with SilverLining

By default, SilverLining will allocate memory using new, delete, malloc, and free. However, you may redirect SilverLining's memory management into your own memory manager if you wish.

The SilverLining::Allocator class is defined in the public header MemAlloc.h. If you extend this class, you may implement your own Allocator::alloc() and Allocator::dealloc() methods to manage memory however you wish. Pass an instantiation of your Allocator-derived class into Allocator::SetAllocator() prior to creating any SilverLining objects, and the SilverLining library and rendering DLL's will use your own allocation scheme.

The Allocator will capture all calls to new, delete, malloc, and free within SilverLining. Every object in SilverLining derives from a SilverLining::MemObject class that overloads the new and delete operators, rather than overloading new and delete globally. Our STL objects also use a custom allocator that routes through SilverLining::Allocator. The macros SL_VECTOR, SL_MAP, SL_LIST, and SL_STRING are used as a convenience for STL objects using our allocator.

7.5 Resource Management with SilverLining

By default, SilverLining will load its texture, data, and shader resources directly from disk relative to the path to the resources directory you specify in Atmosphere::Initialize(). However, all disk access in SilverLining is abstracted by a ResourceLoader class, so you have the ability to hook in any resource management scheme you wish.

See the documentation for ResourceLoader for more details. For example, if you wanted to include all of SilverLining's data, textures, and shaders within your own pack files, you could extend the ResourceLoader class and hook it into your own resource manager. Then, pass a pointer to your derived ResourceLoader into Atmosphere::SetResourceLoader() prior to calling Atmosphere::Initialize(), and all disk access will be routed through your own resource management.

It's important to note that the renderer DLL's for SilverLining also live inside our resource folder, and Windows must load these DLL's directly from disk. So, although you can move all of the other resources in SilverLining to a virtual file system, you do need to retain a resources folder containing the renderer DLL's at least. If you want to avoid this, you can rebuild the SilverLining libraries to statically link in the renderer you want, and eliminate the DLL dependencies altogether. Licensed customers will find build targets in the SilverLining project file to statically link the OpenGL or various DirectX renderers directly into the SilverLining library.

7.6 Stratocumulus Cloud Layers

Take care when using stratocumulus cloud layers in your scene - they are rendered using GPU ray-casting and depend on a sophisticated fragment program. Unlike other cloud layers that will run on pretty much any hardware, stratocumulus clouds do require a system that support Shader Model 3.0, and a newer GPU that has a fast fragment processor. If you're running a simulation on known, modern graphics hardware, stratocumulus cloud layers will reward you with per-fragment lighting and extremely

dense clouds with constant rendering time. However, for consumer applications where the system requirements are less controlled, you may want to stick with cumulus congestus clouds instead.

7.7 Simulating Sand Storms

SilverLining includes a SANDSTORM cloud layer type, used to simulate sand storms, dust storms, or "haboobs".

Sandstorms should have a base altitude equal to your local ground level. They use "soft particles" to avoid artifacts where the cloud puffs intersect the ground at the altitude you specify.

It is not necessary to specify a thickness or density with sandstorms. Here is an example of setting up a 50km x 50km sandstorm positioned 25km from the origin, with a ground altitude at sea level:

```
// Sandstorms should be positioned at ground level. There is no need to set their

// density or thickness.
static void SetupSandstorm()
{
    CloudLayer *sandstormLayer;

    sandstormLayer = CloudLayerFactory::Create(SANDSTORM);
    sandstormLayer->SetIsInfinite(false);
    sandstormLayer->SetLayerPosition(0, -25000);
    sandstormLayer->SetBaseAltitude(0);
    sandstormLayer->SetBaseLength(50000);
    sandstormLayer->SetBaseWidth(50000);
    sandstormLayer->SeedClouds(*atm);

    atm->GetConditions()->AddCloudLayer(sandstormLayer);
}
```

7.8 Integration Tips with Popular Scene Graphs

7.8.1 Integrating with OpenSceneGraph

Silverlining should integrate easily into any rendering engine built on top of OpenGL or DirectX, but we get the most questions about integrating with OpenSceneGraph. We've included some sample code for OpenSceneGraph 2.4 and up with the SDK to get you started; it's a modified osgviewer application that integrates the sky, clouds, and lighting. Also included is an example of using SilverLining with multiple windows in OSG.

A more limited example for OSG 1.0 is also provided.

If you are using osgEarth, getting started is even easier; a sample driver for SilverLining is included with osgEarth, and it can be a quick way to get something up and running quickly. You'll probably want to extend this driver to meet your specific needs however.

We also include a public domain integration of SilverLining with OSG from osgRecipes (https://github.com/xarray/osgRecipes). In many ways, this integration is better than our own - it's simpler, with a single SilverLiningNode. For non-geocentric scenes, it's an easier way to get started. You'll find a geocentric version of this code in the osgEarth example.

Studying or extending this sample code is the easiest way to get started, but if you want to start from scratch, here are the things you need to know about OpenSceneGraph integration:

- The easiest way to ensure Atmosphere::DrawSky and Atmosphere::DrawObjects are called at the right time is to wrap these calls in osg::Drawable objects. Then, create an osg::Geode for each and add the Drawables to them, and set the Geodes into a renderbin that's used with the rest of your scene. Just set the object that calls DrawSky to have an early position in the renderbin (like -1) and the DrawObjects drawable to a very high position.

- If you're going to use multithreaded mode, you need to take care that Atmosphere::Initialize is called in the rendering thread. An easy way to do this is keeping a static boolean in the drawable that calls DrawSky, and call the initialization right before DrawSky is called for the first time. That's also the right time to set up your cloud layers.

- Any call to SilverLining that might result in OpenGL calls needs to happen while a valid GL context is active. This can include seeding cloud layers and removing them, as well as calls that you'd expect to draw the sky and clouds. Make sure these calls are all done within a drawImplementation() method to be safe.

- You'll probably want to disable how OpenSceneGraph automatically computes the near and far clip planes and set these explicitly, since OSG won't know about the bounds of SilverLining's sky box and its clouds. Alternately, you can use the implementation of ClampProjectionMatrixCallback included with the sample code to try and include the bounds of the sky's objects and modify the size of the sky box as necessary. We recommend simply disabling the automatic computation of the clip planes unless you need to do viewpoints from space where it's necessary to push out the near clip plane in order to preserve depth buffer resolution.

- Remember when SilverLining draws the sky, it will clear the color buffer and the depth buffer in the process. You can get a little performance back by disabling buffer clearing in OpenSceneGraph, as it's redundant.

7.8.2 Integrating with Ogre3D

The sample code directory of the SDK includes source code that illustrates Ogre3D integration. It's built on top of a standard Ogre standalone application framework from the Ogre tutorials.

Integration with Ogre3D is fairly straightforward. The trick is to implement a RenderQueueListener in order to get the hooks you need for the beginning and end of frame. The sample code also illustrates how to obtain the pointer to the Direct3D device from Ogre3D, which you'll need if you're using Direct3D instead of OpenGL. Another Direct3D-specific gotcha: you need to tell SilverLining to render its lighting pass offscreen, since Ogre3D's backbuffer is not readable. The sample code shows you how to call Atmosphere::SetConfigOption() to achieve this.

Be sure to set your camera's far clip plane sufficiently far out to encompass our skybox and the clouds.

7.8.3 Integrating with Gamebryo(R) Lightspeed(tm)

An official partner integration of SilverLining with Gamebryo Lightspeed is available separately from http://www.sundog-soft.com or from http://pulse.emergent.net. You'll need to have both the SilverLining SDK and Lightspeed installed before installing the integration package. The integration includes a core runtime library (which may be adapted for use with earlier versions of Gamebryo,) integration with Lightspeed's rapid iteration game framework, and a plug-in for World Builder. Full documentation for Gamebryo integration is included with the integration package.

7.8.4 Integrating with Carmenta Engine

Sample code for integrating SilverLining into a 3D runtime scene powered by the Carmenta geospatial engine is also provided with the SDK.

The main challenge when integrating with Carmenta is that the modelview and projection matrices are not set when the "beforeUpdate" callback is executed. This means the view matrix needs to be constructed from scratch from the 3D view object's properties, and passed to SilverLining. Refer to the sample Carmenta integration code to see how to construct this matrix correctly.

7.8.5 Integrating with NVidia SceniX

We also provide a sample integration with NVidia's SceniX scene engine. You'll find a modified "simple viewer" sample application in the SDK, which uses the SilverLiningManager class included in the sample to wrap your scene's root node with SilverLining's rendering methods, and to keep a directional light object in sync with the simulated conditions.

Examine the SimpleViewerView.cpp file for an example of configuring the atmospheric conditions.

7.9 Using SilverLining in Geocentric or ECEF (Earth Centered Earth Fixed) Coordinates

SilverLining has successfully been integrated into many "whole-Earth" applications, using coordinate systems where the origin is at the center of the Earth, with major axes pointing through the North Pole and through latitude, longitude (0,0). Wrapping your head around how to configure SilverLining's basis vectors and how to position cloud layers in this sort of environment is a bit tricky, so here are some tips.

First, you will need to pass true for the geocentric parameter in Atmosphere::DrawSky() each frame.

Prior to seeding your clouds, and once per frame, you must pass SilverLining new Up and Right vectors such that "up" is the normalized vector from the origin (center of the earth) to your location, and "right" points East from your location. The best way to do this is to construct your "up" vector by normalizing the vector from the center of the Earth to your viewpoint. Construct a "north" vector pointing in the direction of your north pole (ie, 0,0,1) and compute the cross product north X up. Normalize the resulting cross product to create your "right" vector pointing east.

If your Earth model is in a system where Y is North instead of Z, you'll need to set the configuration setting geocentric-z-is-up to "no" in the file resources/silverlining.config. Otherwise, the position of the sun, moon, and stars will be incorrect relative to your skies and your Earth.

To position a localized cloud layer, first set the up and right vectors to reflect the layer's desired position on the Earth, and call SetLayerPosition() with coordinates (0,0). For infinite cloud layers, it's not necessary to set the layer position at all. Silverlining will automatically keep track of where sea level is at the current camera location, and adjust the positions of the clouds to ensure their altitudes remain correct.

Here is some sample code that might make it easier to understand, which creates a cloud layer over the camera position in geocentric space:

```
// cameraPosition is a Vector3 in ECEF coordinates, relative to the
// center of the Earth.
Vector3 up = cameraPosition;
up.Normalize();
Vector3 north(0, 0, 1); // We assume Z is up
Vector3 east = north.Cross(up);
east.Normalize();
atm->SetUpVector(up.x, up.y, up.z);
atm->SetRightVector(east.x, east.y, east.z);

// Note - be sure to also update AtmosphericConditions::SetLocation
// if your position changes, and update SetUpVector and SetRightVector
```

```
// as your position changes each frame.

// Creates a cumulus congestus cloud layer at 2500 meters above the camera.
cumulusCongestusLayer = CloudLayerFactory::Create(CUMULUS_CONGESTUS);
cumulusCongestusLayer->SetBaseAltitude(2500);
cumulusCongestusLayer->SetThickness(100);
cumulusCongestusLayer->SetBaseLength(80000);
cumulusCongestusLayer->SetBaseWidth(80000);
cumulusCongestusLayer->SetDensity(0.4);
cumulusCongestusLayer->SetLayerPosition(0, 0);
cumulusCongestusLayer->SeedClouds(*atm);
atm->GetConditions()->AddCloudLayer(cumulusCongestusLayer);
```

Some whole-earth environments render all objects relative to the current camera position in order to preserve floating point precision. If this applies to your system, the above technique still works, but you'll want to multiply a translation matrix to negate out the camera position into the view matrix passed into Atmosphere::SetCameraMatrix() every frame for the clouds to appear in the correct positions.

In a geocentric / ECEF system, be sure to use Atmosphere::GetSunPositionGeographic(), Atmosphere::GetMoonPositionGeographic(), and/or Atmosphere::GetSunOrMoonPositionGeographic() to retrieve the direction of the sun or moon light in the correct coordinate system. The usual calls to Atmosphere::GetSunPosition() etc. will return horizon coordinates, which isn't what you want.

If you intend to render the Earth from space, be sure to also look at the geocentric parameter of Atmosphere::DrawSky(). At very high altitudes, drawing the sun, moon, and stars using horizon coordinates will break down; this parameter will cause everything to be rendered in geographic coordinates instead. Also check out the enable-atmosphere-from-space setting in resources/SilverLining.config. With this on, SilverLining will render a ring around the Earth representing the atmosphere as you leave it - it assumes that a unit in your coordinate system represents one meter, and you're rendering the Earth itself with a realistic size. Be sure that your near and far clip planes take this ring into account, or you may end up clipping it out - see the projection matrix callback class included in the OpenSceneGraph sample code for an example of how to account for the atmosphere's geometry when setting the near and far clip planes.

We've included some sample code in the osgEarth example project included with the SDK. It illustrates the techniques described above within the context of an OpenSceneGraph / osgEarth application in geocentric space.

7.10 Simulating Precipitation with SilverLining

SilverLining includes particle systems that will simulate any amount of rain, sleet, or snowfall that's likely to occur in nature. It will also pass back fog settings that will let you accurately simulate the reduction in visiblity due to the precipitation. SilverLining takes the precipitation rate you specify, and calculates the distribution of particle sizes, particle velocities, and visiblity using research from the meteorology community based

on real-world observations.

Precipitation may be attached to cloud layers, so as you pass underneath a cloud, the precipitation will automatically start and stop.

Although it's sophisticated, it's easy to use. All you need to do is call CloudLayer::SetPrecipitation() on a stratus or cumulus CloudLayer object. You can do this when you're initializing your CloudLayers, or change it any any time thereafter.

CloudLayer::SetPrecipitation only takes two values - the precipitation type (CloudLayer::RAIN , CloudLayer::SLEET, or CloudLayer::SNOW), and the precipitation rate. The rate is specified in millimeters per hour - in the case of snow, this is the liquid equivalent precipitation rate, and not the accumulation rate. Reasonable values for the rate would range from 1.0 to 30.0.

You may simulate mixed precipitation by calling CloudLayer::SetPrecipitation consecutively with different precipitation types. For example, to simulate a 50/50 blend of sleet and snow at an overall rate of 20 mm/hr, you could call SetPrecipitation(CloudLayer::SLEET, 10.0); SetPrecipitation(CloudLayer::WET_SNOW, 10.0);

If you want to turn precipitation off on a CloudLayer, just call CloudLayer::SetPrecipitation() with a precipitation type of CloudLayer::NONE. Doing this will remove all precipitation effects you set previously on this CloudLayer.

If you want precipitation effects to be applied globally, independently of CloudLayers, you may instead use the AtmosphericConditions::SetPrecipitation() method. This allows you to use SilverLining's precipitation effects without using SilverLining's clouds, if you so desire.

To take advantage of the visibility reduction effects, be sure to implement the code described in "Fog Effects with SilverLining" below.

7.11 Manipulating Time with SilverLining

By default, SilverLining will simulate the time you specify with AtmosphericConditions::SetTime(). Clouds will move with the wind over time, but the sun, moon, stars, and appearance of the sky won't change until you call SetTime() again. This is adequate for most applications, and it ensures good performance.

Some applications need to control the passage of time. You might need to go backwards in time in order to replay a scene. Or you might want to accelerate time for a time-lapse photography sort of effect. Or, you may simply want to filter the time between frames to ensure smooth animation. SilverLining allows you to replace its own internal millisecond timer with your own, to accomplish such effects.

To do so, implement a MillisecondTimer class of your own, and pass it to AtmosphericConditions::SetMillisecondTimer(). Here's an example of a MillisecondTimer that speeds up the passage of time 10X:

```
class MyMillisecondTimer : public MillisecondTimer
{
public:
        virtual unsigned long GetMilliseconds() const
        {
                return timeGetTime() * 10;
        }
};
```

To use this timer, pass it into your AtmosphericConditions class, like this:

```
MyMillisecondTimer *timer = new MyMillisecondTimer();
atm->GetConditions()->SetMillisecondTimer(timer);
```

You are responsible for deleting your timer class at shutdown. To restore the default timer, call SetMillisecondTimer(NULL).

Your new timer will influence the rate at which clouds move, but by default, that's all that millisecond timers influence. You may also set the sun, moon, stars, and appearance of the sky to change dynamically over time. To do so, call AtmosphericConditions::EnableTimePassage(). For example:

```
atm->GetConditions()->EnableTimePassage(true, 20000);
```

The first parameter enables dynamic passage of the simulated time. The second specifies the minimum time interval, in milliseconds (as defined by your own MillisecondTimer if you're using one) between cloud deck relighting passes. Relighting a cloud deck is a relatively expensive operation, and you may not want to relight during an interactive scene. You'll at least want to do so infrequently, as the above example does. If you want the sun, moon, stars, and sky to change smoothly over time but don't want to incur the costs of cloud relighting at all, pass -1 for the second parameter.

One thing to watch out for when accelerating time - if you have cumulonimbus clouds in your scene, this might result in lightning being rendered much more frequently, which can both look strange and affect performance. To counteract this, increase the config setting lightning-max-discharge-period accordingly. Alternately, lightning may be disabled altogether by setting cumulonimbus-lightning-density to 0.

7.12 Considerations for Multi-Channel, Multi-Viewport, and Multi-Context Setups

Many training simulator applications require a consistent scene drawn across several systems, or "channels", together. SilverLining provides two approaches to ensuring the clouds are consistent across multiple computers.

The simplest solution is to seed the standard library's random number generator consistently on each channel, using srand(). Cloud layers generated in the same order with the same parameters will then be consistent.

Alternately, you may use the CloudLayer::Save() method to pre-generate cloud layers for your scene, and save them to disk. Then, you can distribute these saved cloud layers to each channel, and load them at runtime using CloudLayer::Restore(). This approach also has the benefit of speeding up your application's initialization a bit.

Some applications also require drawing multiple scenes on a single channel, across multiple windows or graphics contexts. Other applications render multiple viewports within a single context. Using SilverLining across multiple contexts or viewports in the same application is supported, but requires some care.

If you are drawing across multiple windows that may span different graphics contexts, each viewport must have its own SilverLining::Atmosphere object associated with it. This means that for each viewport, you must instantiate and initialize a separate Atmosphere object, and use the correct Atmosphere for the context you are currently rendering to. The Atmosphere must be initialized while its associated graphics context is currently active, and any cloud layers added to the scene must also be added to each Atmosphere while the associated context is active.

Internally, SilverLining maintains a concept of a "current atmosphere" that is updated whenever Atmosphere::Initialize() or Atmosphere::DrawSky() is called. Any operations to the Atmosphere, such as adding or removing cloud layers, must happen following one of these calls.

So, to recap: if you're rendering across multiple graphics contexts, make sure you associate an Atmosphere object for each viewport. You should only add or remove cloud layers following a call to Atmosphere::Initialize() or Atmosphere::DrawObjects() in this situation, and the Atmosphere and cloud layers must be initialized and drawn while the correct window or graphics context is currently active. Generally, the easiest way to accomplish this is by initializing and modifying the Atmosphere object within your application's drawing method.

The other case is drawing across multiple windows or viewports that share a graphics context, for example using WGL's "context sharing" functionality. In this case, it is safe to re-use the same Atmosphere object across each viewport, taking care to set the new camera and projection matrix prior to drawing the sky and clouds on each viewport. In order to ensure precipitation effects work properly across the different views, make sure to specify the "camera" parameter on the Atmosphere::DrawSky() and Atmosphere::DrawObjects() methods.

7.13 Managing Your Own Drawing

For many applications, allowing SilverLining to draw its own clouds is a simple approach that will get you up and running quickly with good results. Other applications

may prefer to manage their own cloud drawing, in order to ensure that clouds are sorted properly with respect to other translucent objects in the scene. SilverLining provides you with access to the underlying translucent objects it normally renders in Atmosphere::DrawObjects() for this purpose.

If you call Atmosphere::DrawObjects(false), this tells SilverLining that it should build up a list of translucent objects to draw within the DrawObjects call, but not to actually draw them. After calling DrawObjects(false), you may then obtain a list of translucent objects with the Atmosphere::GetObjects() call.

Translucent objects are generally rendered last in a scene, in back-to-front order with respect to the current viewpoint. Use the Atmosphere::GetObjectDistance() method to obtain the distance from a given viewpoint to each object for sorting purposes. Once your translucent objects are sorted and you're ready to draw them, you may draw a SilverLining ObjectHandle using the Atmosphere::DrawObject() method. Note, you must have blending enabled prior to calling DrawObject().

Here's an example of manually drawing clouds following a call to Atmosphere::DrawObjects(false):

```
static bool comp(ObjectHandle c1, ObjectHandle c2)
{
        double d1 = atm->GetObjectDistance(c1, c2, camX, camY, camZ);
        double d2 = atm->GetObjectDistance(c2, c1, camX, camY, camZ);

        return (d1 > d2);
}

void DrawClouds()
{
        SL_VECTOR(ObjectHandle>& objs = atm-)GetObjects();

        sort(objs.begin(), objs.end(), comp);

        glEnable(GL_BLEND);
        glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
        glEnable(GL_DEPTH_TEST);
        glDepthMask(0);
        glEnable(GL_TEXTURE_2D);
        glDisable(GL_LIGHTING);
        glDisable(GL_FOG);

        SL_VECTOR(ObjectHandle)::iterator it;
        for (it = objs.begin(); it != objs.end(); it++)
        {
                atm->DrawObject(*it);
        }

        glDepthMask(1);
}
```

If you instead call DrawObjects(true), this code isn't necessary, and DrawObjects() will draw the cloud objects on its own.

7.14 Crepuscular Rays

SilverLining can automatically render crepuscular rays from the sun when it is occluded by a cloud layer. This effect is also known as "God rays", shafts of light, and volumetric lighting.

To enable crepuscular rays, simply pass true for the crepuscularRays parameter in SilverLining::Atmosphere::DrawObjects(). When enabled, rays will be drawn over the scene when the camera is facing the sun. The intensity of the rays will vary based on the cloud coverage above the camera, so no rays will be drawn if there are no cloud layers present.

Our implementation of crepuscular rays is a postprocessing effect, so it does not have access to depth information from the scene. By default, the rays will render in front of anything drawn before Atmosphere::DrawObjects() is called and in front of the clouds. The SilverLining.config setting "crepuscular-rays-depth" may be used to adjust the depth of the rays in the scene; you may prefer to have them render behind your scene's objects, in which case you would set the depth to 1.

Also bear in mind that SilverLining does not have access to your scene's geometry, so rays will only be cast based on the clouds in the scene. Other objects in your scene that might occlude the sun will not affect the rays.

The overall intensity of the rays may be adjusted with the crepuscular-rays-exposure setting in Resources/SilverLining.config.

7.15 Advanced: using shadow maps

SilverLining makes it easy to cast shadows from the clouds onto your terrain. Just have a look at the SilverLining::Atmosphere::GetShadowMap() method.

When calling GetShadowMap(), SilverLining will render the clouds from the point of view of the sun or moon into a texture map returned to you. Along with this texture map you'll get a matrix that will transform world coordinates into texture coordinates on the shadow map. All you need to do is multiply this texture in while rendering your terrain, while applying the matrix provided to obtain texture coordinates for it. This is straightforward when incorporating the shadow map into a fragment shader.

The best time to call Atmosphere::GetShadowMap() is after calling Atmosphere::DrawObjects(). Avoid calling it in the middle of rendering your scene, and especially in between calls to Atmosphere::DrawSky() and Atmosphere::DrawObjects() - this will require SilverLining to perform an extra culling pass, and hurt performance.

If you're using fixed function pipeline rendering under OpenGL or DirectX 9, here are examples of applying the shadow map to a surface. First, in OpenGL:

```
void *texHandle;
Matrix4 mat, shadowMatrix;
```

```
atm->GetShadowMap(texHandle, &mat, &shadowMatrix, false);
Gluint shadowMap = (GLuint)texHandle;

glActiveTexture(GL_TEXTURE1);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, shadowMap);

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGendv(GL_S, GL_EYE_PLANE, shadowMatrix.GetRow(0));
glEnable(GL_TEXTURE_GEN_S);

glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGendv(GL_T, GL_EYE_PLANE, shadowMatrix.GetRow(1));
glEnable(GL_TEXTURE_GEN_T);
```

The above code configures automatic texture coordinate generation using the matrix provided by Atmosphere::GetShadowMap(), and binds the shadow map to texture unit 1. You'll also need to configure OpenGL to multiply this texture in with your base texture on unit 0, like this:

```
// Stage 1 = prev * shadow
glActiveTexture(GL_TEXTURE1);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glEnable(GL_TEXTURE_2D);

// Stage 0 = tex * primary color
glActiveTexture(GL_TEXTURE0);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glEnable(GL_TEXTURE_2D);
```

Again, this is more flexible if you're using shaders. Here's equivalent code under DirectX 9 - note that you need to multiply in the inverse of the view matrix with DirectX 9 in order to get into world space, since it doesn't do it automatically like OpenGL does.

```
SilverLining::Matrix4 mvp, xform;
void *texture;
if (atm->GetShadowMap(texture, &mvp, &xform, false))
{
    device->SetTexture(1, (IDirect3DTexture9*)texture);

    float mf[16];
    int i = 0;
    for (int row = 0; row < 4; row++)
    {
        for (int col = 0; col < 4; col++)
        {
            mf[i++] = (float)xform.elem[row][col];
        }
    }

    D3DXMATRIX worldToTextureMatrix(mf);

    D3DXMATRIX invView, textureMatrix;
    D3DXMatrixInverse(&invView, NULL, &view);
```

```
    D3DXMatrixMultiply(&textureMatrix, &invView, &worldToTextureMatrix);

    device->SetSamplerState(1, D3DSAMP_ADDRESSU, D3DTADDRESS_WRAP);
    device->SetSamplerState(1, D3DSAMP_ADDRESSV, D3DTADDRESS_WRAP);
    device->SetSamplerState(1, D3DSAMP_BORDERCOLOR, 0xFFFFFFFF);
        device->SetSamplerState(1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
        device->SetSamplerState(1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);

    device->SetTransform(D3DTS_TEXTURE1, &textureMatrix);
    device->SetTextureStageState(1, D3DTSS_TEXCOORDINDEX,
        D3DTSS_TCI_CAMERASPACEPOSITION | 1);
    device->SetTextureStageState(1, D3DTSS_TEXTURETRANSFORMFLAGS, D3DTTFF_PROJECT
      ED | D3DTTFF_COUNT3);

    device->SetTextureStageState(1, D3DTSS_COLOROP,D3DTOP_MODULATE);
    device->SetTextureStageState(1, D3DTSS_COLORARG1,D3DTA_TEXTURE);
    device->SetTextureStageState(1, D3DTSS_COLORARG2,D3DTA_CURRENT);
    device->SetTextureStageState(1, D3DTSS_ALPHAOP,   D3DTOP_SELECTARG1 );
    device->SetTextureStageState(1, D3DTSS_ALPHAARG1, D3DTA_CURRENT );

}
```

Options exist in Atmosphere::GetShadowMap() to control the amount of clouds included in the shadow map, and the darkness of the shadows. See the documentation for the method for more details. If you configure it to render entire cloud layers into the shadow map, you might be able to avoid generating the shadow map each frame - but this will come at the cost of lower-resolution shadows.

7.16 Creating environment maps from SilverLining

SilverLining provides the SilverLining::Atmosphere::GetEnvironmentMap() method to automatically generate a cube map texture that represents the sky and clouds from the view of the current camera position.

By default, each face of the cube map will be 256x256. If you require higher resolution, this may be adjusted in the resources/silverlining.config file under the "environment-map-size" setting.

If there are clouds in your scene, you will need to regenerate your cube map whenever the camera or the clouds move by some significant amount for accurate reflections.

Cube maps generated such as this may be passed into Sundog Software's Triton Ocean SDK for generating water reflections from SilverLining's skies.

7.17 Using SilverLining with High Dynamic Range applications (HDR)

Under the hood, all of SilverLining's lighting is physically simulated and done in units of kilo-candelas per square meter - we then tone-map these raw physical values down

to a displayable range.

This tone-mapping may be disabled by calling Atmosphere::EnableHDR(). This may be used in conjunction with a floating-point render target to receive the sky and clouds rendered in units of kilo-candelas per square meter, instead of [0,1.0]. The user is responsible for performing their own tone mapping on the entire scene in this case, as the floating point render target is transferred to a displayable surface.

When HDR is enabled, lighting values returned such as Atmosphere::GetSunOrMoonColor() will also no longer be tone mapped or clamped.

HDR support requires shader model 3.0 support (since pixel shader inputs will be clamped otherwise.)

If the rest of the scene is not rendered in units of kCD/m^2, you can scale the HDR output to better match the rest of your scene. In the resources/silverlining.config file, look for settings such as sun-transmission-scale and sun-scattered-scale which will multiply our internal lighting values by the given scale factor, thereby brightening or darkening the scene. You may also influence the brightness of the clouds (as opposed to the sky) independently with the setting sun-luminance-scale, which will further modulate the lighting values applied to the clouds.

Scaling the overall brightness of the sky itself may be accomplished with hosek-wilkie-radiance-scale.

As part of disabling tone-mapping, gamma correction will also be disabled in HDR mode. Be sure to account for gamma correction in your own tone-mapping operator to avoid skies that look too dark.

The brightness of lightning in HDR mode may be adjusted using the lightning-hdr-boost config setting.

7.18 Selecting a Sky Model

SilverLining allows you to choose between two different mathematical models for simulating sky colors for a given time and location; each has their own advantages and disadvantages. You may select your sky model using the SilverLining::Atmosphere::SetSkyModel() method. The sky model may be changed at runtime, allowing you to compare the two models we offer side by side.

The PREETHAM sky model is an extension of the Preetham paper from the 1999 SIGGRAPH proceedings. It offers a good mixture of simplicity, speed, and accuracy, and our implementation of it has the benefit of having been refined since SilverLining's initial launch in 2006. We use our own solar radiance model with it, as well as our own model of twilight and nighttime luminance to extend the original algorithms.

The HOSEK_WILKIE sky model is newer, from the 2012 SIGGRAPH proceedings. It is similar to the Preetham approach, but is more complex and handles higher turbidities and colors near the horizon more accurately. You'll find that the Hosek-Wilkie model

produces sky colors that may be more realistic, but are less dramatic at lower turbidity values. Game developers may prefer the PREETHAM model, while simulation developers may prefer HOSEK_WILKIE. Our implementation of Hosek-Wilkie also uses our own solar radiance model and tone-mapping. One limitation of the Hosek-Wilkie approach is that it can only simulate the effects of sunlight when the sun is above the horizon. Due to this, we blend Hosek-Wilkie with our Preetham model implementation as the sun approaches the horizon, in order to retain SilverLining's twilight effects and sky illumination from moonlight.

7.19 Positioning Individual Clouds Within a Cloud Layer

Cumulus congestus and cumulus mediocris cloud layers will, by default, fill the layer with clouds with a realistic size distribution in order to achieve the cloud coverage requested with CloudLayer::SetDensity(). However, the placement of these clouds will be random. If you are attempting to represent something like a frontal boundary or a specific cloud formation, randomly scattered clouds may not be sufficient for your needs.

We offer the SilverLining::CloudLayer::AddCloudAt() method to allow you to place your own clouds in addition to, or instead of, the randomly distributed clouds in the cloud layer. To use this capability, first the seed the cloud layer with any randomly-positioned clouds you want. Even if you want all clouds to be hand-positioned, you must still call CloudLayer::SeedClouds() prior to placing individual clouds (you would just use CloudLayer::SetDensity(0) to prevent random clouds from being created.) Then, for each cloud, call CloudLayer::AddCloudAt() with the position and dimension of each individual cloud. The positions passed in must be relative to the center of the cloud layer, at its base, and represent the position of the center of each cloud's base. You must ensure the positions passed in are within the bounds of the cloud layer as defined by its SetBaseWidth() and SetBaseLength() calls.

Here is an example of setting up a cumulus congestus cloud layer, with 20% coverage from random scattered clouds, and a straight line of 2x2x1 km clouds extending across the center of the layer.

```
cumulusCongestusLayer = CloudLayerFactory::Create(hasCumulusCongestusHiRes ?
  CUMULUS_CONGESTUS_HI_RES : CUMULUS_CONGESTUS);
cumulusCongestusLayer->SetBaseAltitude(4000);
cumulusCongestusLayer->SetThickness(500);
cumulusCongestusLayer->SetBaseLength(80000);
cumulusCongestusLayer->SetBaseWidth(80000);
cumulusCongestusLayer->SetLayerPosition(0, 0);
cumulusCongestusLayer->SetFadeTowardEdges(true);
cumulusCongestusLayer->SetIsInfinite(true);
cumulusCongestusLayer->SetCurveTowardGround(false);

if (cumulusCongestusLayer->SupportsAddCloudAt())
{
    cumulusCongestusLayer->SetDensity(0.2);
```

```
        cumulusCongestusLayer->SeedClouds(*atm);

        for (double x = -40000; x < 40000; x += 5000)
        {
            Vector3 pos(x, 0, 0); // position relative to layer center
            Vector3 size(2000, 1000, 2000); // each cloud is 2km across and 1km h
    igh
            cumulusCongestusLayer->AddCloudAt(*atm, pos, size);
        }
    }
```

Please refer to the documentation for SilverLining::CloudLayer::AddCloudAt() for more information.

7.20 Using SilverLining with Linear Color Space

Some applications render internally in linear color space, which negates the effect of gamma correction in order to improve lighting and shading. If your application does this, you'll probably find that SilverLining's skies appear too bright, since it is applying gamma correction to the sky.

To correct this, use Atmosphere::SetGamma(1.0) - this will disable SilverLining's gamma correction and draw the sky in linear color space instead.

7.21 Tweaking SilverLining's Appearance

SilverLining is highly configurable. Its default settings attempt to accurately simulate the sky for the conditions prescribed, together with tone-mapping to account for human perception and how your eyes adjust to low light levels. But, if you don't like the appearance of the sky, clouds, or precipitation under a given set of circumstances, odds are you can adjust it to your liking. Most of the "knobs and dials" for SilverLining may be found in the resources/SilverLining.config file. Open it up in your development environment and examine the available settings; most are well-documented within the file.

To change a setting, save your changes into the SilverLining.config file, and restart your application. Some changes may be configurable at runtime by using SilverLining::Atmosphere::SetConfigOption().

The config file consists of the following main sections:

- General Settings - These control high-level rendering strategies, the overall tone mapping brightness, and defaults.
- Sky Settings - Controls the appearance of the sky box, and scaling values for the simualted sun and moonlight.

- Precipitation - These settings control the appearance and density of precipitation particles.
- Atmosphere From Space - Controls the size and resolution of the "atmospheric limb" when viewing the atmosphere from space.
- Glare Settings - Controls physically-based glare effects surrounding the sun, moon, and bright stars.
- Lightning Settings - Tweaks the complexity and color of lightning strikes from cumulonimbus clouds.
- Cloud Settings - Adjusts the appearance of cumulus clouds in general.
- Specific Cloud Type Settings - Adjusts the simulated droplet particle properties, cloud growth models, and cloud distribution models for each cloud type.

For example, if you want to increase the brightness of the clouds overall, adjust the setting ambient-scattering up. For deeper shadows within the clouds, adjust cumulus-lighting-quick-and-dirty-attenuation down.

The overall brightness of the sky and light may be influenced with the "brightness" setting.

For richer colors in the atmosphere, experiment with the other air-mass-model settings (ICQ or NREL.) You can also adjust the air mass directly using the air-mass-multiplier setting; higher air masses result in more atmospheric scattering.

The color of sunsets and sunrises may also be directly adjusted, using the settings sunset-X-boost, sunset-Y-boost, sunset-Z-boost, and sunset-boost-exponent. To make the sunsets redder, increase the sunset-X-boost a bit. These settings represent a boost applied to the scattered sunlight color in XYZ color space. The exponent controls how these boosts are applied as the sun's angle from the horizon changes. Higher integers for sunset-boost-exponent will limit the effect of the boost to be closer to the horizon; lower values will make the boosts more pervasive.

For more visible rain particles, increase rain-streak-brightness and rain-streak-width-multiplier.

The color and density of the fog inside stratus clouds may be adjusted using the stratus-fog-$*$ settings.

Want a bigger sun or moon? Increase sun-width-degrees or moon-width-degrees.

If you wish to keep your changes to SilverLining.config separate from the default configuration file, you may keep your changes in a SilverLining.override file placed alongside SilverLining.config. This way, you can update to SilverLining.config files in new versions of SilverLining without worrying about merging your changes back in.

Many other tweaks are possible; examine the config file for more options, and feel free to contact us at support@sundog-soft.com if you have any questions about them.

Chapter 8

Class Index

8.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

SilverLining::Allocator 59
SilverLining::LightningListener 154
SilverLining::MemObject 177
SilverLining::Atmosphere 60
SilverLining::AtmosphericConditions 104
SilverLining::CloudLayer 118
SilverLining::CloudLayerFactory 143
SilverLining::Color 146
SilverLining::Frustum 151
SilverLining::LocalTime 155
SilverLining::Location 165
SilverLining::Matrix3 169
SilverLining::Matrix4 173
SilverLining::Plane 180
SilverLining::Vector3 187
SilverLining::Vector3f 192
SilverLining::Vector4 195
SilverLining::Vertex 198
SilverLining::WindVolume 200
SilverLining::MillisecondTimer 179
SilverLining::RandomNumberGenerator 183
SilverLining::ResourceLoader 184

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

SilverLining::Allocator (You may extend the Allocator class to hook your own memory management scheme into SilverLining) 59
SilverLining::Atmosphere (This class is the main interface to SilverLining) . 60
SilverLining::AtmosphericConditions (A class to set the current simulation conditions for the sky) 104
SilverLining::CloudLayer (A collection of clouds of uniform classification organized into a "deck" of similar altitudes) 118
SilverLining::CloudLayerFactory (A class factory for CloudLayer objects) . 143
SilverLining::Color (A RGBA color, where each component is represented as a float from 0-1) 146
SilverLining::Frustum (A simple class that just collects six planes together and calls it a frustum) 151
SilverLining::LightningListener (Extend this class in order to receive notifications of lightning strike events from cumulonimbus cloud layers) 154
SilverLining::LocalTime (A class to represent the simulated local time, time zone, and daylight savings time observation) 155
SilverLining::Location (An object representing a geographic location) 165
SilverLining::Matrix3 (A simple 3x3 matrix class and its operations) 169
SilverLining::Matrix4 (An implementation of a 4x4 matrix and some simple operations on it) 173
SilverLining::MemObject (This base class for all SilverLining objects intercepts the new and delete operators, routing them through SilverLining::Allocator()) 177

SilverLining::MillisecondTimer (Exposes an interface to allow applications to control millisecond-level time) 179
SilverLining::Plane (A geometric plane, modelled as a normal vector and a distance to the plane) 180
SilverLining::RandomNumberGenerator (Extend this class in order to use your own random number generator with SilverLining) 183
SilverLining::ResourceLoader (This class is used whenever SilverLining needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish) 184
SilverLining::Vector3 (A 3D double-precision Vector class and its operations) 187
SilverLining::Vector3f (A 3D single-precision vector class, and its operations) 192
SilverLining::Vector4 (A simple double-precision 4D vector class with no operations defined) 195
SilverLining::Vertex (A single vertex containing a position, RGBA color, and 3D texture coordinates) 198
SilverLining::WindVolume (Defines an area of a given wind velocity and direction bounded by two altitudes) 200

Chapter 10

File Index

10.1 File List

Here is a list of all documented files with brief descriptions:

C:/sl/Public Headers/Atmosphere.h (The main interface to SilverLining) . . . 207
C:/sl/Public Headers/AtmosphericConditions.h (Configures the time, location, atmosphere, and cloud cover) . . . 209
C:/sl/Public Headers/CloudLayer.h (Methods for configuration and initializing cloud decks) . . . 210
C:/sl/Public Headers/CloudLayerFactory.h (Instantiates specific types of CloudLayer objects) . . . 211
C:/sl/Public Headers/CloudTypes.h (Defines the enumerated values of cloud layer types that may be passed to CloudLayerFactory::Create) . . . 212
C:/sl/Public Headers/Color.h (A class that defines an RGBA color and operations on it) . . . 214
C:/sl/Public Headers/Frustum.h (A viewing frustum, modeled as a collection of six planes) . . . 215
C:/sl/Public Headers/LightningListener.h (A virtual base class for receiving lightning strike events in your application) . . . 217
C:/sl/Public Headers/LocalTime.h (An object to describe the local time, time zone, and daylight savings time observation) . . . 218
C:/sl/Public Headers/Location.h (Represents the geographic location being simulated) . . . 221
C:/sl/Public Headers/Matrix3.h (Implements a 3x3 matrix and its operations) 223
C:/sl/Public Headers/Matrix4.h (An implementation of a 4x4 matrix and some simple operations on it) . . . 224
C:/sl/Public Headers/MemAlloc.h (Memory allocation interface for SilverLining) . . . 226

C:/sl/Public Headers/MillisecondTimer.h (Exposes an interface applications may use to manipulate time) 228
C:/sl/Public Headers/Plane.h (A class that models a geometric plane and its operations) 229
C:/sl/Public Headers/RandomNumberGenerator.h (A virtual base class for overriding SilverLining's random number generator in your application) 231
C:/sl/Public Headers/ResourceLoader.h (A class for loading SilverLining's resources from mass storage, which you may extend) 233
C:/sl/Public Headers/SilverLiningDLLCommon.h (Shared header for the specific renderer DLL's, and for use by the Renderer class of the engine) 234
C:/sl/Public Headers/Vector3.h (A 3D Vector class and its operations) 236
C:/sl/Public Headers/Vector4.h (A simple 4D vector class) 237
C:/sl/Public Headers/WindVolume.h (Defines an area of a given wind speed and direction) 239

Chapter 11

Class Documentation

11.1 SilverLining::Allocator Class Reference

You may extend the Allocator class to hook your own memory management scheme into SilverLining.

`#include <MemAlloc.h>`

Collaboration diagram for SilverLining::Allocator:

Public Member Functions

- virtual void ∗SILVERLINING_API alloc (size_t bytes)

  Allocate a block of memory; defaults to malloc()

- virtual void SILVERLINING_API dealloc (void ∗p)

  Free a block of memory; defaults to free()

Static Public Member Functions

- static Allocator *SILVERLINING_API GetAllocator ()

  Retrieves the static allocator object.

- static void SILVERLINING_API SetAllocator (Allocator *a)

  Sets a new static allocator object.

Static Protected Member Functions

- static Allocator *SILVERLINING_API GetDefaultAllocator ()

  Retrieves default allocator object.

11.1.1 Detailed Description

You may extend the Allocator class to hook your own memory management scheme into SilverLining. Instantiate your own implementation of Allocator, and pass it into Allocator::SetAllocator prior to calling any other SilverLining methods or instantiating any SilverLining objects.

Each object in SilverLining overloads the new and delete operators, and routes memory management through the Allocator as well.

11.1.2 Member Function Documentation

11.1.2.1 static void SILVERLINING_API SilverLining::Allocator::SetAllocator (Allocator * *a*)

`[inline, static]`

Sets a new static allocator object.

If this is not called, the default implementation using malloc and free is used.

The documentation for this class was generated from the following file:

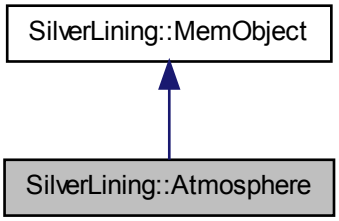
- C:/sl/Public Headers/MemAlloc.h

11.2 SilverLining::Atmosphere Class Reference

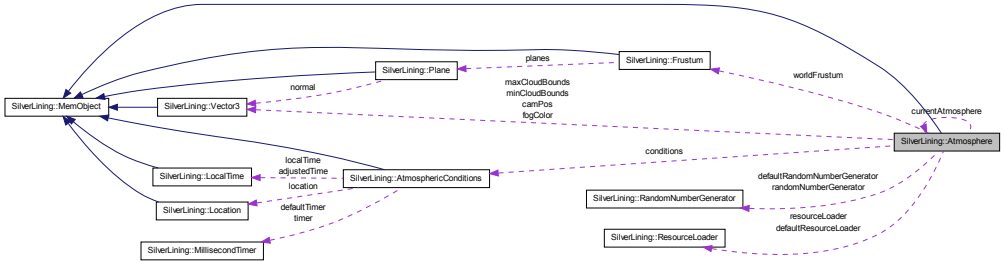
This class is the main interface to SilverLining.

```
#include <Atmosphere.h>
```

Inheritance diagram for SilverLining::Atmosphere:



Collaboration diagram for SilverLining::Atmosphere:



Public Member Functions

- Atmosphere (const char ∗userName, const char ∗licenseKey)

  Constructor.

- ∼Atmosphere ()

  Destructor.

- void SILVERLINING_API SetConditions (const AtmosphericConditions &conditions)

  Configures the simulated cloud, wind, time, and location.

- const AtmosphericConditions &SILVERLINING_API GetConditions () const

  Accessor to the current cloud, wind, time, and location settings.

- void SILVERLINING_API SetSkyModel (SkyModel skyModel)

  Sets the physical model used for simulating sky colors.

- SkyModel SILVERLINING_API GetSkyModel () const

  Retrieves the current sky model in use.

- void SILVERLINING_API GetSunOrMoonPosition (float *x, float *y, float *z) const

  Retrieve the normalized direction of the dominant directional light source.

- void SILVERLINING_API GetSunOrMoonPositionGeographic (float *x, float *y, float *z) const

  Retrieve the normalized direction of the dominant directional light source.

- void SILVERLINING_API GetSunOrMoonPositionEquatorial (float *x, float *y, float *z) const

  Retrieve the normalized direction of the dominant directional light source.

- void SILVERLINING_API GetSunPosition (float *x, float *y, float *z) const

  Retrieve the normalized direction of the sun's light source.

- void SILVERLINING_API GetMoonPosition (float *x, float *y, float *z) const

  Retrieve the normalized direction of the moon's light source.

- void SILVERLINING_API GetSunPositionGeographic (float *x, float *y, float *z) const

  Retrieve the direction of the sun, relative to the center of the Earth in a geocentric coordinate system, where the z axis points from the center of the Earth to the North Pole, and x points toward the prime meridian.

- void SILVERLINING_API GetMoonPositionGeographic (float *x, float *y, float *z) const

  Retrieve the direction of the moon, relative to the center of the Earth in a geocentric coordinate system, where the z axis points from the center of the Earth to the North Pole, and x points toward the prime meridian.

- void SILVERLINING_API GetSunPositionEquatorial (float *x, float *y, float *z) const

  Retrieve the direction of the sun in equatorial coordinates; where x points toward the vernal equinox (where the equator and ecliptic intersect), z points through the north pole.

- void SILVERLINING_API GetMoonPositionEquatorial (float *x, float *y, float *z) const

  Retrieve the direction of the moon in equatorial coordinates; where x points toward the vernal equinox (where the equator and ecliptic intersect), z points through the north pole.

- void SILVERLINING_API GetSunOrMoonColor (float *r, float *g, float *b) const

  Returns the color of the dominant directional light source.

- void SILVERLINING_API GetSunColor (float *r, float *g, float *b) const

  Returns the color of the sun's light source.

- void SILVERLINING_API GetMoonColor (float *r, float *g, float *b) const

  Returns the color of the moon's light source.

- void SILVERLINING_API GetAmbientColor (float *r, float *g, float *b) const

  Returns the color of the ambient "skylight".

- void SILVERLINING_API GetHorizonColor (float yawDegrees, float pitchDegrees, float *r, float *g, float *b) const

  Returns the average color of the sky at the horizon.

- void SILVERLINING_API GetHorizonColor (float pitchDegrees, float *r, float *g, float *b) const

  Returns the average color of the sky at the horizon.

- void SILVERLINING_API GetZenithColor (float *r, float *g, float *b) const

  Returns the color of the sky at the zenith.

- bool SILVERLINING_API GetFogEnabled () const

  Returns whether SilverLining would like to suggest fog settings.

- void SILVERLINING_API GetFogSettings (float *density, float *r, float *g, float *b) const

  Returns exponential fog settings appropriate for your scene.

- void SILVERLINING_API SetHaze (float hazeR, float hazeG, float hazeB, double hazeDepth, double hazeDensity)

  Causes the sky to blend toward a specified "haze color" toward the horizon.

- void SILVERLINING_API GetHaze (float &hazeR, float &hazeG, float &hazeB, double &hazeDepth, double &hazeDensity)

Retrieves the haze parameters set previously by SetHaze().

- void SILVERLINING_API SetGamma (double gamma)

  Sets the value used for gamma correction of the display.

- double SILVERLINING_API GetGamma () const

  Retrieves the value being used for display gamma correction.

- void SILVERLINING_API ForceLightingRecompute ()

  Force SilverLining to recompute all cloud lighting.

- AtmosphericConditions *SILVERLINING_API GetConditions ()

  Return a reference to the current simulated conditions.

- int SILVERLINING_API Initialize (int renderer, const char *resourceDirectoryPath, bool rightHanded, void *environment)

  Call this immediately after constructing your scene's Atmosphere and initializing your graphics subsystem (OpenGL, OpenGL32, DirectX9, DirectX10, DirectX11, or DirectX11.1).

- int SILVERLINING_API Initialize (int renderer, const char *resourceDirectoryPath, bool rightHanded, void *environment, const SL_VECTOR(unsigned int)&userShaders)

  OpenGL-only variant of Atmosphere::Initialize() that takes in a list of user-compiled shader objects that will be linked into all subsequently linked shader program objects.

- void SILVERLINING_API SetUpVector (double x, double y, double z)

  Sets the assumption of what direction is "up".

- void SILVERLINING_API GetUpVector (double &x, double &y, double &z)

  Returns the direction that SilverLining assumes is "up" as a unit vector.

- void SILVERLINING_API SetRightVector (double x, double y, double z)

  Sets the assumption of what direction is "right".

- void SILVERLINING_API GetRightVector (double &x, double &y, double &z)

  Returns the direction that SilverLining assumes is "right" as a unit vector.

- void SILVERLINING_API SetCameraMatrix (const double *cameraMatrix)

  Call this at the beginning of each rendering loop, prior to calling DrawSky().

- void SILVERLINING_API SetProjectionMatrix (const double *projectionMatrix)

Call this at the beginning of each rendering loop, prior to calling DrawSky().

- void SILVERLINING_API SetViewport (int x, int y, int w, int h)

  Call this at the beginning of each rendering loop, prior to calling DrawSky().

- void SILVERLINING_API SetDepthRange (float nearDepth, float farDepth)

  Call this at the beginning of each rendering loop, prior to calling DrawSky(), indicating the range of depth buffer values in use.

- const double *SILVERLINING_API GetCameraMatrix () const

  Returns an array of 16 doubles representing the view matrix last set via SetCameraMatrix().

- const double *SILVERLINING_API GetProjectionMatrix () const

  Returns an array of 16 doubles representing the projection matrix last set via SetProjectionMatrix().

- bool SILVERLINING_API GetViewport (int &x, int &y, int &w, int &h) const

  Returns the viewport size and position last set via SetViewport().

- void SILVERLINING_API UpdateSkyAndClouds ()

  Explicitly updates the ephemeris model and cloud shapes and positions.

- void SILVERLINING_API CullObjects (bool threadSafe=true)

  Explicity performs culling tests against the view frustum for the clouds in the scene.

- bool SILVERLINING_API DrawSky (bool drawSky, bool geocentricMode=false, double skyBoxDimension=0, bool drawStars=true, bool clearDepth=true, bool drawSunAndMoon=true, CameraHandle camera=0)

  Call this at the beginning of your rendering loop.

- bool SILVERLINING_API DrawObjects (bool drawClouds=true, bool drawPrecipitation=true, bool enableDepthTest=true, float crepuscularRays=0.0f, bool enableDepthWrites=false, CameraHandle camera=0, bool backFaceCullClockWise=true, bool drawBackdrops=true)

  Call this at the end of your rendering loop.

- bool SILVERLINING_API GetShadowMap (void *&texture, SilverLining::Matrix4 *lightViewProjMatrix, SilverLining::Matrix4 *worldToShadowMapTexCoord, bool wholeLayers=true, float shadowDarkness=1.0f, bool moonShadows=false, double maxShadowObjectDistance=-1.0)

  Creates and retrieves a shadow map containing the clouds in the scene, the view / projection matrix used to create it, and the matrix you'll need to map it to your terrain.

- bool SILVERLINING_API GetEnvironmentMap (void *&texture, int facesToRender=6, bool floatingPoint=false, CameraHandle cameraID=0, bool drawClouds=true, bool drawSunAndMoon=true)

  Generates and retrieves a cube map texture representing the sky as viewed from the current camera position (as specified by the last call to Atmosphere::SetCameraMatrix().) This can be useful for rendering reflections, for example with Sundog Software's Triton Ocean SDK.

- void SILVERLINING_API OverrideCloudLighting (float ambientR, float ambientG, float ambientB, float diffuseR, float diffuseG, float diffuseB)

  Use this to use specific colors for lighting the clouds, instead of the sun, moon, and sky colors.

- void SILVERLINING_API ClearCloudLightingOverride ()

  Clears any manual overrides of the cloud light color that were previously set with Atmosphere::OverrideCloudLighting().

- double SILVERLINING_API GetFramerate () const

  Return the current framerate, based on the rendering time of the previous frame, in frames per second.

- SL_VECTOR (ObjectHandle)&SILVERLINING_API GetObjects() const

  If you called DrawObjects() with the drawClouds parameter set to false, no clouds will actually be drawn.

- void SILVERLINING_API ClearObjects ()

  Clears the internal list of objects retrieved by GetObjects().

- float SILVERLINING_API GetObjectDistance (ObjectHandle obj, float obj2X, float obj2Y, float obj2Z, float sortPosX, float sortPosY, float sortPosZ)

  If you are managing your own cloud drawing, you'll need to sort them from back to front along with the other objects in your scene.

- float SILVERLINING_API GetObjectDistance (ObjectHandle obj, ObjectHandle obj2, float sortPosX, float sortPosY, float sortPosZ)

  If you are managing your own cloud drawing, you'll need to sort them from back to front along with the other objects in your scene.

- void SILVERLINING_API GetObjectPosition (ObjectHandle obj, float &x, float &y, float &z)

  Obtains the position, in world coordinates, of a given ObjectHandle obtained from the list returned from GetObjects().

- void SILVERLINING_API DrawObject (ObjectHandle obj)

  If you are managing your own cloud object drawing by calling DrawObjects(false), then your sorted list of objects may drawn by calling DrawObject() on each object.

- void SILVERLINING_API ReloadConfigFile ()

  Reloads the SilverLining.config file in the resources directory.

- void SILVERLINING_API SetConfigOption (const char *key, const char *value)

  Sets a specific SilverLining.config value.

- const char *SILVERLINING_API GetConfigOptionString (const char *key) const

  Retrieves a specific SilverLining.config value as a string.

- double SILVERLINING_API GetConfigOptionDouble (const char *key) const

  Retrieves a specific SilverLining.config value as a double.

- bool SILVERLINING_API GetConfigOptionBoolean (const char *key) const

  Retrieves a specific SilverLining.config value as a double.

- void SILVERLINING_API UpdateEphemeris ()

  Forces the ephemeris model to recalculate the position of the sun, moon, and stars for the currently simulated time and location.

- void SILVERLINING_API SetInfraRedMode (bool bInfraRed)

  Sets infrared sensor simulator mode.

- bool SILVERLINING_API GetInfraRedMode () const

  Gets whether infrared sensor mode is enabled.

- void SILVERLINING_API DisableFarCulling (bool bDisable)

  Allows you to disable culling of clouds against the far clip plane of your projection matrix.

- bool SILVERLINING_API GetFarCullingDisabled () const

  Retrieves whether culling against the far clip plane for clouds is disabled.

- void SILVERLINING_API GetCloudBounds (double &minX, double &minY, double &minZ, double &maxX, double &maxY, double &maxZ)

  Retrieves the bounds of any clouds currently in the scene.

- void SILVERLINING_API EnableLensFlare (bool bEnabled)

Enable or disable a big, flashy lens flare effect when the sun is visible in the scene.

- bool SILVERLINING_API GetLensFlareEnabled () const

  Returns if lens flare is currently enabled.

- void SILVERLINING_API EnableSunOcclusion (bool bEnabled)

  Enable or disable occlusion testing of the sun, required when using GetSunOcclusion().

- bool SILVERLINING_API GetSunOcclusionEnabled () const

  Returns if sun occlusion testing is currently enabled.

- float SILVERLINING_API GetSunOcclusion ()

  Returns the percentage of the sun that is currently visible in the scene, as a float from 0 to 1.0.

- void SILVERLINING_API SetRandomNumberGenerator (const RandomNumberGenerator *rng)

  If you wish to override SilverLining's default random number generator, which is based on the standard library's rand() function, you may do so by extending the RandomNumberGenerator base class, instantiating your extended class, and passing in a pointer to your class here.

- bool SILVERLINING_API Serialize (std::ostream &stream)

  Flatten the Atmosphere and everything in it to a stream - useful for capturing the state of everything to save to disk or for recreating the Atmosphere in response to a device reset in DX9.

- bool SILVERLINING_API Unserialize (std::istream &stream)

  Restore the Atmosphere from a stream generated from Atmosphere::Serialize.

- bool SILVERLINING_API IsInitialized () const

  Returns whether the Initialize() method has been successfully called on this Atmosphere yet, and it is safe to use for rendering tasks.

- void SILVERLINING_API SetSunAlpha (double alpha)

  Sets a transparency value for the sun billboard, useful for fading the sun out with haze.

- double SILVERLINING_API GetSunAlpha () const

  Retrieves the alpha value of the sun, as set by Atmosphere::SetSunAlpha().

- void SILVERLINING_API SetMoonAlpha (double alpha)

Sets a transparency value for the moon billboard, useful for fading the moon out with haze.

- double SILVERLINING_API GetMoonAlpha () const

  Retrieves the alpha value of the moon, as set by Atmosphere::SetMoonAlpha().

- double SILVERLINING_API GetSkyCoverage () const

  Retrieve the percentage of sky occluded by clouds, above the current camera position.

- SL_VECTOR (unsigned int) SILVERLINING_API GetActivePlanarCloudShaders()

  OpenGL only: retrieve a list of OpenGL shader program objects for all currently active planar cloud shaders.

- unsigned int SILVERLINING_API GetBillboardShader () const

  OpenGL only: retrieve the OpenGL shader program object for drawing billboards, including cumulus cloud puffs, the sun, and the moon.

- unsigned int SILVERLINING_API GetSkyShader () const

  OpenGL only: retrieve the OpenGL shader program object for drawing the sky box.

- unsigned int SILVERLINING_API GetStarShader () const

  OpenGL only: retrieve the OpenGL shader program object for drawing the stars.

- unsigned int SILVERLINING_API GetPrecipitationShader () const

  OpenGL only: retrieve the OpenGL shader program object for drawing precipitation particles.

- void SILVERLINING_API ReloadShaders (const SL_VECTOR(unsigned int)&userShaders)

  OpenGL only: recompile and link all shader programs on demand.

- void SILVERLINING_API ForceSunAndMoon (double sunLat, double sunLon, double moonLat, double moonLon)

  Force ecliptic (not horizon) latitude and longitude of the sun & moon to specified positions.

- void SILVERLINING_API ForceSunAndMoon (double sunLat, double sunLon, double moonLat, double moonLon, double moonPhase)

  Force ecliptic (not horizon) latitude and longitude of the sun & moon to specified positions, and force a moon phase.

- void SILVERLINING_API ForceSunAndMoonHorizon (double sunAltitude, double sunAzimuth, double moonAltitude, double moonAzimuth)

Force the sun and moon positions to given horizon coordinates.

- void SILVERLINING_API ForceSunAndMoonHorizon (double sunAltitude, double sunAzimuth, double moonAltitude, double moonAzimuth, double moonPhase)

  Force the sun and moon positions to given horizon coordinates, and force a moon phase.

- void SILVERLINING_API ForceMoonPhase (double moonPhase)

  Force a specific moon phase regardless of the simulated date.

- void SILVERLINING_API ClearForcedMoonPhase ()

  Clear any forced moon phase set with ForceSunAndMoon(), ForceSunAndMoonHorizon(), or ForceMoonPhase()

- void SILVERLINING_API ClearForcedSunAndMoon ()

  Clear any forced sun & moon position set with ForceSunAndMoon()

- bool SILVERLINING_API ReloadCumulusTextures ()

  Reloads the textures used to represent cumulus cloud puffs.

- void SILVERLINING_API SetOutputScale (float scale)

  Scales all final color values by some factor; useful if you need to match SilverLining to some other tone mapping scheme or just want a quick way to darken or lighten everything.

- float SILVERLINING_API GetOutputScale () const

  Retrieve the output scale previously set via SetOutputScale, or 1.0 by default.

- void SILVERLINING_API D3D9DeviceLost ()

  Call when a Direct3D9 device is lost.

- void SILVERLINING_API D3D9DeviceReset ()

  Call when a Direct3D9 device is reset.

Static Public Member Functions

- static void SILVERLINING_API EnableHDR (bool hdr)

  Enabling High Dynamic Range (HDR) mode will disable all tone mapping, clamping, and gamma correction in SilverLining.

- static bool SILVERLINING_API GetHDREnabled ()

Retrieve whether high dynamic range rendering mode is currently enabled.

- static void SILVERLINING_API SetResourceLoader (ResourceLoader *loader)

  Overrides the default FILE - based resource loader with a customer-supplied resource loader object.*

- static ResourceLoader *SILVERLINING_API GetResourceLoader ()

  *Gets the ResourceLoader object being used to load resources, which may be the default FILE * based loader or a customer-supplied loader.*

- static void SILVERLINING_API SetWorldUnits (double meters)

  By default, SilverLining assumes that one world unit is equal to one meter - that is, all of the dimensions for cloud layers, altitudes, etc.

- static double SILVERLINING_API GetUnitScale ()

  Retrieves the scale factor for world units, as set by SetWorldUnits().

11.2.1 Detailed Description

This class is the main interface to SilverLining. The Atmosphere simulates the sky, clouds, and weather.

To use an Atmosphere class, simply instantiate one using the default constructor, and then call the Initialize() method to set it up for either the OpenGL or DirectX rendering subsystems.

You may initialize the atmospheric conditions (cloud decks, wind, time, location, etc.) by populating an AtmosphericConditions class and passing this to the SetConditions method prior to rendering.

Then, within your main rendering loop, as soon as you have set the view and projection matrices for your scene to reflect the current camera position, call the DrawSky() method. This will draw the background of the sky and perform any lighting computations required for the atmosphere.

At this point, you may query GetSunOrMoonPosition() and GetSunOrMoonColor() to obtain the tone-mapped directional lighting information for your outdoor scene, and GetAmbientColor() to get the tone-mapped ambient skylight for your scene.

Finally, at the end of your rendering loop, call DrawObjects(). This will draw all of the clouds within the scene in back-to-front order.

11.2.2 Constructor & Destructor Documentation

11.2.2.1 SilverLining::Atmosphere::Atmosphere (const char * *userName,* const char * *licenseKey*)

Constructor.

Creates an atmosphere object with default settings. You must specify the user name and license key provided with your SilverLining license purchase in order to instantiate an Atmosphere. An invalid name / key combination will result in a dialog box informing the user that this software is unlicensed, and in the application terminating after two minutes of use.

Warning! Displaying a dialog box while in full-screen mode will cause some DirectX applications to crash. To prevent the licensing dialog box on unlicensed SDK's from causing a crash, instantiate your Atmosphere object before entering full-screen mode.

Parameters

userName	The user name, exactly as provided to you in the licensing information received with your SilverLining license purchase. A null-terminated C string.
licenseKey	The license key code, exactly as provided to you with your license purchase. Null-terminated C string.

11.2.2.2 SilverLining::Atmosphere::~Atmosphere ( )

Destructor.

This will clean up all cloud objects owned by the atmospheric conditions.

11.2.3 Member Function Documentation

11.2.3.1 void SILVERLINING_API SilverLining::Atmosphere::ClearCloudLightingOverride ( )

Clears any manual overrides of the cloud light color that were previously set with Atmosphere::OverrideCloudLighting().

11.2.3.2 void SILVERLINING_API SilverLining::Atmosphere::ClearObjects ( )

Clears the internal list of objects retrieved by GetObjects().

11.2.3.3 void SILVERLINING_API SilverLining::Atmosphere::CullObjects (bool *threadSafe* = `true`)

Explicity performs culling tests against the view frustum for the clouds in the scene.

Calling this method is optional; if it is not called prior to Atmosphere::DrawSky(), it will be called automatically. However, this exposes the ability to perform culling in a separate pass or thread from the actual drawing of the clouds. Make sure Atmosphere::SetCameraMatrix() and Atmosphere::SetProjectionMatrix() have been called prior to calling this method, so its view frustum is accurate.

If you are calling Atmosphere::DrawObjects() more than once per frame, be sure to call Atmosphere::CullObjects() prior to each DrawObjects() call.

Parameters

threadSafe	whether a mutex will be enforced to protect against culling from occurring simultaneously with drawing. Needed only in multi-threaded applications.

11.2.3.4 void SILVERLINING_API SilverLining::Atmosphere::DisableFarCulling (bool *bDisable*) `[inline]`

Allows you to disable culling of clouds against the far clip plane of your projection matrix.

This might be useful if you have a very close far clip plane and cannot push it out to encompass the clouds due to depth buffer precision issues. Note, this call will disable culling, not clipping. You'll need to disable clipping explicitly prior to calling Atmosphere::DrawObjects() for this call to have an effect. In OpenGL, you can do this using glEnable(GL_DEPTH_CLAMP_NV) in conjuction with glDepthFunc(GL_LEQUAL).

11.2.3.5 void SILVERLINING_API SilverLining::Atmosphere::DrawObject (ObjectHandle *obj*)

If you are managing your own cloud object drawing by calling DrawObjects(false), then your sorted list of objects may drawn by calling DrawObject() on each object.

Your rendering state must be set for rendering translucent objects. Specifically, you must enable blending with a blend equation of ONE, INVSRCALPHA. Lighting must be off, depth reads enabled, depth writes disabled, fog must be off, and 2D texturing enabled.

Parameters

obj	An object handle obtained from the list returned by GetObjects(), after calling DrawObjects(false).

See also

DrawObjects()
GetObjects()
GetObjectDistance()

11.2.3.6 bool SILVERLINING_API SilverLining::Atmosphere::DrawObjects (bool *drawClouds* = `true`, bool *drawPrecipitation* = `true`, bool *enableDepthTest* = `true`, float *crepuscularRays* = `0.0f`, bool *enableDepthWrites* = `false`, CameraHandle *camera* = `0`, bool *backFaceCullClockWise* = `true`, bool *drawBackdrops* = `true`)

Call this at the end of your rendering loop.

After drawing all of your scene's objects, call the DrawObjects() method. This will draw all of the scene's clouds from back to front, if the drawClouds parameter is set to true. If there are translucent objects in your scene that should be drawn in front of the clouds, you'll want to draw those after calling DrawObjects().

Alternately, you may obtain handles to each cloud object independently and sort them with the other translucent objects in your scene. If you set drawClouds to false, DrawObjects() will build up a list of translucent objects to render for this frame, but not actually render them. You may then access each individual cloud object with the GetObjects() method, sort them against your other translucent objects using the GetObjectDistance() method to obtain their sort distances, and then actually draw them using DrawObject().

For applications where the scene is generally beneath the clouds, setting drawClouds to true and then drawing your own translucent objects after DrawObjects() is generally adequate.

In cases where you call DrawObjects() more than once per frame, be sure to explicitly call Atmosphere::CullObjects() prior to each call.

DrawObjects() must be called from the same thread and render context that called Atmosphere::Initialize(). It should be balanced with a call to DrawSky() prior to calling DrawObjects().

Parameters

drawClouds	Set to true if you want DrawObjects to actually draw all of the clouds, sorted from back to front. Set to false if you just want to build up a list of clouds to draw but not actually draw them; this list may be obtained with GetObjects().
drawPrecipitation	Set to false in order to disable rendering of precipitation effects.

enableDepthTest	By default, translucent objects in the scene will read from the depth buffer but not write to them. If you have objects behind your clouds, this ensures they are drawn properly. However, in some situations, you may have a ground-based viewpoint where nothing is behind the clouds, and you may not have a depth buffer at all, or a depth buffer with undefined contents. In these situations, you may want to disable depth reads by setting enableDepthTest to false.
crepuscularRays	Set to a value greater than 0 to draw crepuscular rays (AKA "God rays") after the clouds, rendering shafts of light through the clouds. This results in an extra rendering pass, so does come at some performance cost. The value controls the intensity of the effect; 1.0 will result in the default intensity (as configured in SilverLining.config under crepuscular-rays-exposure), while values less than 1.0 will scale the effect down. To minimize the performance cost of crepuscular rays, disable lens flare effects using the disable-lens-flare setting in the resource/silverlining.config file - the two effects are costly when used together.
enableDepthWrites	Normally you would not want transparent objects such as clouds to write to the depth buffer, but there are some specialized situations, such as writing to a depth buffer for shadows, where this might be needed. Setting this to true will cause the clouds to write to the depth buffer; it is false by default.
camera	An optional pointer to your own camera object, or some sort of identifier for the camera currently being drawn. This is useful if you are using the same Atmosphere objects to draw the sky on multiple viewports or windows that share the same graphics context. By specifying a camera, we can associate precipitation particle positions with each one, allowing precipitation effects to work properly across multiple views with the same Atmosphere. If you are using separate Atmosphere objects per view, have only one view, or are not using precipitation effects, there is no need to set this.
backFaceCullClockWise	Determines whether back-face culling should cull out clockwise-winding polygons, which is normally the default. If you are doing something like rendering a reflection texture where the up axis is flipped, then you also need to flip the winding order for backface culling, and setting this to false allows you to do that.
drawBackdrops	If 2D dynamically-updated imposter backdrops are in use for representing distant clouds, this parameter allows you to skip drawing them. Useful for preserving performance and consistency when drawing views where distant clouds don't matter much, such as reflection passes.

Returns

true if the clouds were successfully drawn.

11.2.3.7 bool SILVERLINING_API SilverLining::Atmosphere::DrawSky (bool *drawSky*, bool *geocentricMode* = `false`, double *skyBoxDimension* = `0`, bool *drawStars* = `true`, bool *clearDepth* = `true`, bool *drawSunAndMoon* = `true`, CameraHandle *camera* = `0`)

Call this at the beginning of your rendering loop.

At the start of each frame in your scene, first call SetCameraMatrix() and SetProjectionMatrix(). Then, call DrawSky(). This will draw the skybox for the simulated time and location, including the sun, moon, stars, and planets, and perform any necessary lighting calculations.

DrawSky() should be called from the same rendering context and thread that was active when Atmosphere::Initialize() was called. Otherwise, you may experience a crash inside this method.

Alternately, DrawSky() may be called instead at the end of your rendering loop (prior to drawing translucent objects and prior to calling DrawObjects()) if you set the clearDepth parameter to false. This may result in better performance, as it reduces overdraw against the sky box by rendering the sky box last instead of first.

It is important that the camera and projection matrices are set to reflect your scene's current camera prior to calling DrawSky(). Versions of SilverLining prior to 1.7 would extract these matrices automatically, but beginning with 1.7 you MUST call SetCameraMatrix() and SetProjectionMatrix(). We made this change in order to support DirectX 10, which has no fixed function pipeline for us to query for these matrices.

Every call to DrawSky() should be balanced with a later call to DrawObjects() or GetObjects().

Parameters

drawSky	Pass false to suppress drawing of the skybox, but still perform any lighting calculations required.
geocentricMode	If set to true, the stars, planets, sun, and moon will be drawn in a geocentric coordinate frame, where the X axis points through the prime meridian, Z points through the North Pole, and the origin is at the center of the Earth. This is generally only useful for space-based viewpoints. Normally, you'll leave this set to false so that astronomical objects will be drawn in local horizon coordinates.
skyBoxDimension	Sets an explicit dimension in world units for the length of a face of the sky box. Useful to prevent clipping of the skybox in systems that dynamically adjust the near and far clip planes; you can specify a value each frame that fits within them. Most applications can just leave this set to 0, in which case it will draw the sky box with the default size specified in SilverLining.config (1000)
drawStars	Set true to draw the stars when the sky is dark, and conditions are not overcast - or false to explicitly disable the drawing of the stars under all circumstances, such as in the presence of thick fog.

clearDepth	Set true to clear depth buffer contents whith sky, false to not clear depth and draw sky with Z test (useful for front to back sorting).
drawSunAndMoon	Set true to enable drawing of the sun and moon, or false to disable them (for exmaple, if you're drawing your own sun and moon billboards, or you're in heavy fog and the sun and moon should not be visible at all.)
camera	An optional pointer to your own camera object, or some sort of identifier for the camera currently being drawn. This is useful if you are using the same Atmosphere objects to draw the sky on multiple viewports or windows that share the same graphics context. By specifying a camera, we can associate precipitation particle positions with each one, allowing precipitation effects to work properly across multiple views with the same Atmosphere. If you are using separate Atmosphere objects per view, have only one view, or are not using precipitation effects, there is no need to set this.

Returns

true if the skybox was successfully drawn.

11.2.3.8 static void SILVERLINING_API SilverLining::Atmosphere::EnableHDR (bool *hdr*)

`[inline, static]`

Enabling High Dynamic Range (HDR) mode will disable all tone mapping, clamping, and gamma correction in SilverLining.

This is only useful if you are rendering to a floating point framebuffer and applying a tone mapping operator of your own as a postprocessing step - otherwise, it'll just have the effect of making the sky and clouds pure white during the day.

The values written to the framebuffer in HDR mode will represent kilo-candelas per square meter, which may or may not be on a luminance scale comparable to the rest of your scene. The config settings sun-transmission-scale, sun-scattered-scale, moon-transmission-scale, and moon-scattered-scale can be used to scale our luminance values into a range comparable with your own. (You'll find these settings in the file resources/silverlining.config).

HDR is a static state; its current setting will apply to all Atmosphere objects at once. Under DirectX, HDR requires shader model 3.0 support.

See also

GetHDREnabled()

Parameters

hdr	True to enable high dynamic range mode and disable all tone mapping and gamma correction. False to tone-map the sky to the displayable range of [0,1.0]. HDR is disabled by default.

11.2.3.9 void SILVERLINING_API SilverLining::Atmosphere::EnableLensFlare (bool *bEnabled*) [inline]

Enable or disable a big, flashy lens flare effect when the sun is visible in the scene.

11.2.3.10 void SILVERLINING_API SilverLining::Atmosphere::EnableSunOcclusion (bool *bEnabled*) [inline]

Enable or disable occlusion testing of the sun, required when using GetSunOcclusion().

Disable this if you will never use GetSunOcclusion() to regain performance when the sun is in view.

11.2.3.11 void SILVERLINING_API SilverLining::Atmosphere::ForceLightingRecompute ( ) [inline]

Force SilverLining to recompute all cloud lighting.

Under normal operation, SilverLining only recomputes cloud lighting when the simulated time, location, or cloud cover changes, or when the camera moves a significant amount relative to each cloud. If you wish to force a lighting computation for all clouds to take place on the next frame, call this method.

There is a performance impact for calling this. Under normal circumstances, there is no reason to call this method.

11.2.3.12 void SILVERLINING_API SilverLining::Atmosphere::ForceMoonPhase (double *moonPhase*)

Force a specific moon phase regardless of the simulated date.

Moon phase ranges from 0 (new moon) to 1.0 (full moon.) Call Atmosphere::ClearForcedMoonPhase() to return to automatic computation of the phase.

11.2.3.13 void SILVERLINING_API SilverLining::Atmosphere::ForceSunAndMoon (double *sunLat,* double *sunLon,* double *moonLat,* double *moonLon,* double *moonPhase*)

Force ecliptic (not horizon) latitude and longitude of the sun & moon to specified positions, and force a moon phase.

Note latitudes and longitudes are in radians, and the moon phase ranges from 0 (new moon) to 1.0 (full moon.) Call Atmosphere::ClearForcedSunAndMoon() and Atmosphere::ClearForcedMoonPhase() to return to automatic computation based on the time and location.

11.2.3.14 void SILVERLINING_API SilverLining::Atmosphere::ForceSunAndMoon (double *sunLat,* double *sunLon,* double *moonLat,* double *moonLon*)

Force ecliptic (not horizon) latitude and longitude of the sun & moon to specified positions.

Note latitudes and longitudes are in radians. Call Atmosphere::ClearForcedSunAndMoon() to return to automatic computation based on the time and location.

11.2.3.15 void SILVERLINING_API SilverLining::Atmosphere::ForceSunAndMoonHorizon (double *sunAltitude,* double *sunAzimuth,* double *moonAltitude,* double *moonAzimuth*)

Force the sun and moon positions to given horizon coordinates.

Call Atmosphere::ClearForcedSunAndMoon() to return to automatic computation based on time and location. Altitudes and azimuths are all in radians.

11.2.3.16 void SILVERLINING_API SilverLining::Atmosphere::ForceSunAndMoonHorizon (double *sunAltitude,* double *sunAzimuth,* double *moonAltitude,* double *moonAzimuth,* double *moonPhase*)

Force the sun and moon positions to given horizon coordinates, and force a moon phase.

Call Atmosphere::ClearForcedSunAndMoon() and Atmosphere::ClearForcedMoonPhase() to go back to automatic computation based on time and location. Altitudes and azimuths are in radians, and the moon phase ranges from 0 (new moon) to 1.0 (full moon.)

11.2.3.17 void SILVERLINING_API SilverLining::Atmosphere::GetAmbientColor (float * *r,* float * *g,* float * *b*) const

Returns the color of the ambient "skylight".

This color is suitable for use as an ambient light color for your scene. When used together with the directional light position and color returned by GetSunOrMoonPosition and GetSunOrMoonColor, an accurate simulated model of natural lighting conditions may be obtained.

The color returned by this method is arrived at by simulating the light scattered over the sky above your simulated location at your simulated time, commonly referred to as "skylight." This light is then tone-mapped to account for high dynamic range in the same manner as the directional light. This color is also affected by the presence of thick cloud decks above the camera position.

Parameters

r	A pointer to a float to receive the red channel of the ambient color, from 0 -1.0.
g	A pointer to a float to receive the green channel of the ambient color, from 0 -1.0.
b	A pointer to a float to receive the blue channel of the ambient color, from 0 -1.0.

11.2.3.18 unsigned int SILVERLINING_API SilverLining::Atmosphere::GetBillboardShader ( ) const

OpenGL only: retrieve the OpenGL shader program object for drawing billboards, including cumulus cloud puffs, the sun, and the moon.

11.2.3.19 const double* SILVERLINING_API SilverLining::Atmosphere::GetCameraMatrix ( ) const [inline]

Returns an array of 16 doubles representing the view matrix last set via SetCameraMatrix().

11.2.3.20 void SILVERLINING_API SilverLining::Atmosphere::GetCloudBounds (double & *minX*, double & *minY*, double & *minZ*, double & *maxX*, double & *maxY*, double & *maxZ*)

Retrieves the bounds of any clouds currently in the scene.

Parameters

minX	The minimum x coordinate of the bounding box enclosing all of the clouds.
minY	The minimum y coordinate of the bounding box enclosing all of the clouds.
minZ	The minimum z coordinate of the bounding box enclosing all of the clouds.
maxX	The maximum x coordinate of the bounding box enclosing all of the clouds.
maxY	The maximum y coordinate of the bounding box enclosing all of the clouds.
maxZ	The maximum z coordinate of the bounding box enclosing all of the clouds.

11.2.3.21 const AtmosphericConditions& SILVERLINING_API SilverLining::Atmosphere::GetConditions ( ) const

Accessor to the current cloud, wind, time, and location settings.

Returns a const reference to an AtmosphericConditions class that contains the settings for the current simulation.

11.2.3.22 AtmosphericConditions* SILVERLINING_API SilverLining::Atmosphere::GetConditions ( ) [inline]

Return a reference to the current simulated conditions.

Use this to change the current simulated time, location, etc., by manipulating the AtmosphericConditions object owned by the Atmosphere class directly.

11.2.3.23 bool SILVERLINING_API SilverLining::Atmosphere::GetConfigOptionBoolean (const char * *key*) const

Retrieves a specific SilverLining.config value as a double.

11.2.3.24 double SILVERLINING_API SilverLining::Atmosphere::GetConfigOptionDouble (const char * *key*) const

Retrieves a specific SilverLining.config value as a double.

11.2.3.25 const char* SILVERLINING_API SilverLining::Atmosphere::GetConfigOptionString (const char * *key*) const

Retrieves a specific SilverLining.config value as a string.

11.2.3.26 bool SILVERLINING_API SilverLining::Atmosphere::GetEnvironmentMap (void *& *texture*, int *facesToRender* = 6, bool *floatingPoint* = false, CameraHandle *cameraID* = 0, bool *drawClouds* = true, bool *drawSunAndMoon* = true)

Generates and retrieves a cube map texture representing the sky as viewed from the current camera position (as specified by the last call to Atmosphere::SetCameraMatrix().) This can be useful for rendering reflections, for example with Sundog Software's Triton Ocean SDK.

The size of each cube map face texture is determined by the SilverLining.config setting "environment-map-size", and defaults to 256x256 per face.

This method ends up rendering the sky and clouds six times, so avoid calling this every frame. It will impact performance if you do.

Parameters

texture	A pointer to the platform-specific texture handle containing the cube map. On OpenGL this is a GLuint; on DirectX9 it is a LPDIRECT3DCUBETEXTURE9; on DirectX10 it is a ID3D10ShaderResourceView *; on DirectX11 it is a ID3D11ShaderResourceView *.

facesToRender	The number of cube map faces to render in this call. Normally you'll leave this at the default of 6 to generate a complete cube map, but if you want to spread out the load of generating the cube map over multiple frames, you could set this to something lower. For example, you could call GetEnvironmentMap() with facesToRender set to 1 and call it six times to update the cube map over six calls. We keep track of the last face rendered, and loop through them sequentially each time GetEnvironmentMap is called.
floatingPoint	If true, the environment will be rendered to a 16-bit-per-channel floating point texture on each cube map face. If false, an 8-bit-per-channel RGBA texture will be created instead.
cameraID	An optional pointer to an object representing your application's camera that you want to create an environment map for. Use of this parameter allows you to maintain separate environment maps for individual cameras.
drawClouds	Whether clouds should be drawn into the environment map. If you need to speed up this call and don't care about cloud reflections, this could be a useful option for you.
drawSunAndMoon	Whether the sun and moon should be drawn into the environment map. If you're using this environment map for use on reflections, sometimes the sun or moon reflection can interfere with specular reflections and you might want this option on.

Returns

True if the environment map was successfully generated.

11.2.3.27 bool SILVERLINING_API SilverLining::Atmosphere::GetFarCullingDisabled ( ) const [inline]

Retrieves whether culling against the far clip plane for clouds is disabled.

See also

DisableFarCulling()

11.2.3.28 bool SILVERLINING_API SilverLining::Atmosphere::GetFogEnabled ( ) const

Returns whether SilverLining would like to suggest fog settings.

SilverLining depends on fog effects to simulate being inside a stratus or broken stratus cloud deck, or to simulate reduced visibility due to the presence of rain or snow at the camera's location. If this method returns true, then you should call GetFogSettings in order to help determine the appropriate fog configuration for your scene in order to preserve these atmospheric effects.

11.2.3.29 void SILVERLINING_API SilverLining::Atmosphere::GetFogSettings (float * *density*, float * *r*, float * *g*, float * *b*) const

Returns exponential fog settings appropriate for your scene.

If GetFogEnabled() returns true, then call GetFogSettings() to obtain suggested fog values required to preserve in-cloud effects and precipitation effects that depend on fog. The r, g, and b colors returned are the color of the fog itself modulated by the directional light color.

Parameters

density	A fog density appropriate for use in GL_FOG_DENSITY in exponential fog mode. ie, $f = e^{-(density \cdot z)}$
r	A pointer to a float to receive the red component of the fog color, from 0 - 1.0.
g	A pointer to a float to receive the green component of the fog color, from 0 - 1.0.
b	A pointer to a float to receive the blue component of the fog color, from 0 - 1.0.

11.2.3.30 double SILVERLINING_API SilverLining::Atmosphere::GetFramerate ( ) const [inline]

Return the current framerate, based on the rendering time of the previous frame, in frames per second.

11.2.3.31 double SILVERLINING_API SilverLining::Atmosphere::GetGamma ( ) const

Retrieves the value being used for display gamma correction.

See also

SetGamma()

11.2.3.32 void SILVERLINING_API SilverLining::Atmosphere::GetHaze (float & *hazeR*, float & *hazeG*, float & *hazeB*, double & *hazeDepth*, double & *hazeDensity*)

Retrieves the haze parameters set previously by SetHaze().

See SetHaze() for a description of the parameters.

11.2.3.33 static bool SILVERLINING_API SilverLining::Atmosphere::GetHDREnabled ( )

`[inline, static]`

Retrieve whether high dynamic range rendering mode is currently enabled.

See also

EnableHDR()

11.2.3.34 void SILVERLINING_API SilverLining::Atmosphere::GetHorizonColor (float *yawDegrees*, float *pitchDegrees*, float * *r*, float * *g*, float * *b*) const

Returns the average color of the sky at the horizon.

Based on the specified view direction and field of view, this method will return the average color of the sky in the scene at the horizon. This is often an effective choice of a fog color for your scenes, since it will blend well with the sky in the distance - thereby covering up outdoor scenes that lack sufficient terrain to extend to the horizon. Even for scenes that do render to the horizon, this is a good color for fog just for simulating haze and atmospheric perspective.

Note that during sunrise and sunset when a red glow surround the sun, this color may vary widely depending on the yawDegrees parameter.

Parameters

yawDegrees	The camera's yaw value, in degrees east from north. The horizon color will be calculated by averaging the field of view's horizon color about this direction.
pitchDegrees	The number of degrees above the horizon to sample the color from.
r	A pointer to a float to receive the red channel of the horizon color, from 0 - 1.0
g	A pointer to a float to receive the red channel of the horizon color, from 0 - 1.0
b	A pointer to a float to receive the red channel of the horizon color, from 0 - 1.0

11.2.3.35 void SILVERLINING_API SilverLining::Atmosphere::GetHorizonColor (float *pitchDegrees*, float * *r*, float * *g*, float * *b*) const

Returns the average color of the sky at the horizon.

Based on the current camera orientation and field of view, this method will return the average color of the sky in the scene at the horizon. This is often an effective choice of

a fog color for your scenes, since it will blend well with the sky in the distance - thereby covering up outdoor scenes that lack sufficient terrain to extend to the horizon. Even for scenes that do render to the horizon, this is a good color for fog just for simulating haze and atmospheric perspective.

Note that during sunrise and sunset when a red glow surround the sun, this color may vary widely depending on the orientation of the camera.

Parameters

pitchDegrees	The number of degrees above the horizon to sample the color from.
r	A pointer to a float to receive the red channel of the horizon color, from 0 - 1.0
g	A pointer to a float to receive the red channel of the horizon color, from 0 - 1.0
b	A pointer to a float to receive the red channel of the horizon color, from 0 - 1.0

11.2.3.36 bool SILVERLINING_API SilverLining::Atmosphere::GetInfraRedMode ( ) const [inline]

Gets whether infrared sensor mode is enabled.

11.2.3.37 bool SILVERLINING_API SilverLining::Atmosphere::GetLensFlareEnabled ( ) const [inline]

Returns if lens flare is currently enabled.

See also

EnableLensFlare()

11.2.3.38 double SILVERLINING_API SilverLining::Atmosphere::GetMoonAlpha ( ) const

Retrieves the alpha value of the moon, as set by Atmosphere::SetMoonAlpha().

Defaults to 1.0.

11.2.3.39 void SILVERLINING_API SilverLining::Atmosphere::GetMoonColor (float * *r*, float * *g*, float * *b*) const

Returns the color of the moon's light source.

This method will return a color suitable for lighting your scene based on the natural lighting conditions. The value is tone-mapped, so the high dynamic range between night and day and they way the human eye perceives it is modeled. For example, at night it may be suprisingly bright if a full moon is out. This scattering is sensitive to the turbidity setting specified in the AtmosphericConditions class.

This method only simulates natural light sources passing through the atmosphere: the sun, moon, starlight, planetary light, airglow, galactic light, and zodiacal light. At night, it may be appropriate to add in a little extra to simulate city lights for urban or suburban scenes.

If your camera position is below a thick, infinite stratus cloud deck with coverage of 1.0, the effects on lighting will be simulated as well.

Parameters

r	A pointer to a float that will be populated with the red directional light color, ranging from 0 - 1.0.
g	A pointer to a float that will be populated with the green directional light color, ranging from 0 - 1.0.
b	A pointer to a float that will be populated with the blue directional light color, ranging from 0 - 1.0.

11.2.3.40 void SILVERLINING_API SilverLining::Atmosphere::GetMoonPosition (float * *x*, float * *y*, float * *z*) const

Retrieve the normalized direction of the moon's light source.

This method will return a normalized direction vector pointing to the moon, which may be below the horizon.

11.2.3.41 void SILVERLINING_API SilverLining::Atmosphere::GetMoonPositionEquatorial (float * *x*, float * *y*, float * *z*) const

Retrieve the direction of the moon in equatorial coordinates; where x points toward the vernal equinox (where the equator and ecliptic intersect), z points through the north pole.

11.2.3.42 void SILVERLINING_API SilverLining::Atmosphere::GetMoonPositionGeographic (float * *x*, float * *y*, float * *z*) const

Retrieve the direction of the moon, relative to the center of the Earth in a geocentric coordinate system, where the z axis points from the center of the Earth to the North Pole, and x points toward the prime meridian.

11.2.3.43 float SILVERLINING_API SilverLining::Atmosphere::GetObjectDistance (ObjectHandle *obj*, float *obj2X*, float *obj2Y*, float *obj2Z*, float *sortPosX*, float *sortPosY*, float *sortPosZ*)

If you are managing your own cloud drawing, you'll need to sort them from back to front along with the other objects in your scene.

This method will return the screen depth for sorting purposes.

Parameters

obj	An object handle obtained from the list returned by GetObjects(), after calling DrawObjects(false).
obj2X	The X coordinate of the object you are currently comparing this one against.
obj2Y	The Y coordinate of the object you are currently comparing this one against.
obj2Z	The Z coordinate of the object you are currently comparing this one against.
sortPosX	The X coordinate of the viewpoint location you are sorting translucent objects against.
sortPosY	The Y coordinate of the viewpoint location you are sorting translucent objects against.
sortPosZ	The Z coordinate of the viewpoint location you are sorting translucent objects against.

Returns

The screen depth in eye coordinates at the intersection point to this object, taking into account internal depth biases.

See also

DrawObjects()
DrawObject()
GetObjects()

11.2.3.44 float SILVERLINING_API SilverLining::Atmosphere::GetObjectDistance (ObjectHandle *obj*, ObjectHandle *obj2*, float *sortPosX*, float *sortPosY*, float *sortPosZ*)

If you are managing your own cloud drawing, you'll need to sort them from back to front along with the other objects in your scene.

This method will return the screen depth for sorting purposes.

Parameters

obj	An object handle obtained from the list returned by GetObjects(), after calling DrawObjects(false).
obj2	The object you are comparing this object against.

sortPosX	The X coordinate of the viewpoint location you are sorting translucent objects against.
sortPosY	The Y coordinate of the viewpoint location you are sorting translucent objects against.
sortPosZ	The Z coordinate of the viewpoint location you are sorting translucent objects against.

Returns

The screen depth in eye coordinates, taking into account internal depth biases

See also

DrawObjects()
DrawObject()
GetObjects()

11.2.3.45 void SILVERLINING_API SilverLining::Atmosphere::GetObjectPosition (ObjectHandle *obj*, float & *x*, float & *y*, float & *z*)

Obtains the position, in world coordinates, of a given ObjectHandle obtained from the list returned from GetObjects().

11.2.3.46 float SILVERLINING_API SilverLining::Atmosphere::GetOutputScale ( ) const [inline]

Retrieve the output scale previously set via SetOutputScale, or 1.0 by default.

11.2.3.47 unsigned int SILVERLINING_API SilverLining::Atmosphere::GetPrecipitationShader ( ) const

OpenGL only: retrieve the OpenGL shader program object for drawing precipitation particles.

11.2.3.48 const double* SILVERLINING_API SilverLining::Atmosphere::GetProjectionMatrix ( ) const [inline]

Returns an array of 16 doubles representing the projection matrix last set via SetProjectionMatrix().

11.2.3.49 static ResourceLoader* SILVERLINING_API SilverLining::Atmosphere::GetResourceLoader ( ) [inline, static]

Gets the ResourceLoader object being used to load resources, which may be the default FILE * based loader or a customer-supplied loader.

11.2.3.50 void SILVERLINING_API SilverLining::Atmosphere::GetRightVector (double & *x*, double & *y*, double & *z*)

Returns the direction that SilverLining assumes is "right" as a unit vector.

11.2.3.51 bool SILVERLINING_API SilverLining::Atmosphere::GetShadowMap (void *& *texture*, SilverLining::Matrix4 * *lightViewProjMatrix*, SilverLining::Matrix4 * *worldToShadowMapTexCoord*, bool *wholeLayers* = `true`, float *shadowDarkness* = `1.0f`, bool *moonShadows* = `false`, double *maxShadowObjectDistance* = `-1.0`)

Creates and retrieves a shadow map containing the clouds in the scene, the view / projection matrix used to create it, and the matrix you'll need to map it to your terrain.

The texture returned will be an RGBA texture that may be multiplied with your terrain textures to simulate shadows case from the clouds. The shadow map represents an orthographic view from the sun or moon's direction, with clouds rendered as the current ambient color against a white background. The color in the shadow map represents the opacity of the cloud as seen from the ground to the sun or moon with ambient light added in, resulting in soft shadow edges. Unlike other shadow maps, this is not a depth texture - it's meant to be multiplied into your terrain as-is.

Cirrus and cirrocumulus clouds will not cast shadows, but any cumulus or stratus layer will.

The shadow map resolution may be configured with the config setting shadow-map-texture-size, and defaults to 1024x1024.

Calling GetShadowMap() in between calls to DrawSky() and DrawObjects() is not recommended, as it will trigger an additional culling pass you wouldn't otherwise need.

Note that this method may restore the active view and projection matrices to those passed in via SetCameraMatrix() and SetProjectionMatrix(). OpenGL and DirectX9 users may want to push and pop the active matrices surrounding this call to be safe.

Parameters

texture	A pointer to the platform-specific texture handle containing the 2D shadow map. On OpenGL this is a GLuint; on DirectX9 it is a IDirect3DTexture9 *; on DirectX10 it is a ID3D10ShaderResourceView *; on DirectX11 it is a ID3D11ShaderResourceView *.
lightView-ProjMatrix	The view x projection matrix used when rendering the shadow map.

worldToShadowMapTexCoord	The matrix to transform world coordinates into texture map coordinates of the shadow map texture. This is the lightViewProjMatrix multiplied by a matrix to scale and translate from clip space into texture coordinate space for this Atmosphere's rendering system.
wholeLayers	If set to true, the shadow map will encompass the entirety of any cloud layers above the camera's view frustum. This will allow you to re-use this shadow map from frame to frame, by multiplying in a translation matrix to account for any wind movement, until the lighting or cloud conditions in your scene changes. This makes this call more expensive since it must draw every cloud in the layer, but allows you to potentially avoid calling it every frame. If set to false, the shadow map will only contain clouds that affect the current view frustum, and must be regenerated whenever the camera moves. In this mode, the shadow map may make better use of its available resolution if the far clip plane is relatively close to the camera, but it may not work well with distant far clip planes. If you are simulating real-time cloud growth using CloudLayer::SetCloudAnimationEffects(), you will want to create a new shadow map each frame regardless for accurate shadows. If you're not sure, set this to true.
shadowDarkness	This controls how dark the shadows in the shadow map are. If set to the default of 1.0, shadows will take on the color of the tone-mapped ambient light in the scene. If set to zero, the shadows will be black.
moonShadows	If false (the default value,) shadows will only be generated from the sun,. If true, shadows will be generated from either the sun or the moon depending on which is brighter at the time. However, shadows may suddenly switch direction when the moon overtakes the sun's brightness. If the sun or moon is too close to the horizon, shadows will be faded out to prevent anomalies.
maxShadowObjectDistance	If set to a positive value, no objects farther than this distance from the camera will cast shadows. This may be used to tighten up the shadow map's resolution by eliminating distant shadows. If you are using infinite cloud layers that don't extend to the far clip plane, and you're using "false" for the "wholeLayers" parameter, you may need to set this to the maximum distance clouds will extend from the viewpoint in order to ensure the shadow map covers the area underneath the viewpoint. Otherwise, shadows will be centered about the centroid of the view frustum.

Returns

true if a shadow map was successfully returned, false if an error was encountered or you tried to call GetShadowMap() in between calls to DrawSky() and DrawObjects().

11.2.3.52 double SILVERLINING_API SilverLining::Atmosphere::GetSkyCoverage ( ) const [inline]

Retrieve the percentage of sky occluded by clouds, above the current camera position.

11.2.3.53 SkyModel SILVERLINING_API SilverLining::Atmosphere::GetSkyModel ( ) const

Retrieves the current sky model in use.

See also

SetSkyModel().

Returns

One of PREETHAM or HOSEK_WILKIE.

11.2.3.54 unsigned int SILVERLINING_API SilverLining::Atmosphere::GetSkyShader ( ) const

OpenGL only: retrieve the OpenGL shader program object for drawing the sky box.

11.2.3.55 unsigned int SILVERLINING_API SilverLining::Atmosphere::GetStarShader ( ) const

OpenGL only: retrieve the OpenGL shader program object for drawing the stars.

11.2.3.56 double SILVERLINING_API SilverLining::Atmosphere::GetSunAlpha ( ) const

Retrieves the alpha value of the sun, as set by Atmosphere::SetSunAlpha().

Defaults to 1.0.

11.2.3.57 void SILVERLINING_API SilverLining::Atmosphere::GetSunColor (float * *r*, float * *g*, float * *b*) const

Returns the color of the sun's light source.

This method will return a color suitable for lighting your scene based on the natural lighting conditions. The value is tone-mapped, so the high dynamic range between night and day and they way the human eye perceives it is modeled. At sunset or sunrise, there may be an orange, pink, or reddish hue due to the scattering of sunlight through the atmosphere - which is also simulated. This scattering is sensitive to the turbidity setting specified in the AtmosphericConditions class.

If your camera position is below a thick, infinite stratus cloud deck with coverage of 1.0, the effects on lighting will be simulated as well.

Parameters

r	A pointer to a float that will be populated with the red directional light color, ranging from 0 - 1.0.
g	A pointer to a float that will be populated with the green directional light color, ranging from 0 - 1.0.
b	A pointer to a float that will be populated with the blue directional light color, ranging from 0 - 1.0.

11.2.3.58 float SILVERLINING_API SilverLining::Atmosphere::GetSunOcclusion ( )

Returns the percentage of the sun that is currently visible in the scene, as a float from 0 to 1.0.

This can be used for determining if you're in a shadow, or for implementing your own lens flare effects. If you will never use this method, call Atmosphere::EnableSunOcclusion(false) to regain some performance.

11.2.3.59 bool SILVERLINING_API SilverLining::Atmosphere::GetSunOcclusionEnabled ( ) const [inline]

Returns if sun occlusion testing is currently enabled.

See also

EnableSunOcclusion()

11.2.3.60 void SILVERLINING_API SilverLining::Atmosphere::GetSunOrMoonColor (float * *r*, float * *g*, float * *b*) const

Returns the color of the dominant directional light source.

This method will return a color suitable for lighting your scene based on the natural lighting conditions. The value is tone-mapped, so the high dynamic range between night and day and they way the human eye perceives it is modeled. For example, at noon this value is likely white, but at night it may be suprisingly bright if a full moon is out. At sunset or sunrise, there may be an orange, pink, or reddish hue due to the scattering of sunlight through the atmosphere - which is also simulated. This scattering is sensitive to the turbidity setting specified in the AtmosphericConditions class.

This method only simulates natural light sources passing through the atmosphere: the sun, moon, starlight, planetary light, airglow, galactic light, and zodiacal light. At night, it may be appropriate to add in a little extra to simulate city lights for urban or suburban scenes.

If your camera position is below a thick, infinite stratus cloud deck with coverage of 1.0, the effects on lighting will be simulated as well.

Parameters

r	A pointer to a float that will be populated with the red directional light color, ranging from 0 - 1.0.
g	A pointer to a float that will be populated with the green directional light color, ranging from 0 - 1.0.
b	A pointer to a float that will be populated with the blue directional light color, ranging from 0 - 1.0.

11.2.3.61 void SILVERLINING_API SilverLining::Atmosphere::GetSunOrMoonPosition (float * *x*, float * *y*, float * *z*) const

Retrieve the normalized direction of the dominant directional light source.

This method will return a normalized direction vector pointing to the dominant light source, which may be the moon after sunset.

11.2.3.62 void SILVERLINING_API SilverLining::Atmosphere::GetSunOrMoonPositionEquatorial (float * *x*, float * *y*, float * *z*) const

Retrieve the normalized direction of the dominant directional light source.

This method will return a normalized direction vector pointing to the dominant light source, which may be the moon after sunset. The direction is in equatorial coordinates; where x points toward the vernal equinox (where the equator and ecliptic intersect), z points through the north pole.

11.2.3.63 void SILVERLINING_API SilverLining::Atmosphere::GetSunOrMoonPositionGeographic (float * *x*, float * *y*, float * *z*) const

Retrieve the normalized direction of the dominant directional light source.

This method will return a normalized direction vector pointing to the dominant light source, which may be the moon after sunset. The direction is relative to the center of the Earth in a geocentric coordinate system, where the z axis points from the center of the Earth to the North Pole, and x points toward the prime meridian.

11.2.3.64 void SILVERLINING_API SilverLining::Atmosphere::GetSunPosition (float * *x*, float * *y*, float * *z*) const

Retrieve the normalized direction of the sun's light source.

This method will return a normalized direction vector pointing to the sun, which may be below the horizon.

11.2.3.65 void SILVERLINING_API SilverLining::Atmosphere::GetSunPositionEquatorial (float * *x*, float * *y*, float * *z*) const

Retrieve the direction of the sun in equatorial coordinates; where x points toward the vernal equinox (where the equator and ecliptic intersect), z points through the north pole.

11.2.3.66 void SILVERLINING_API SilverLining::Atmosphere::GetSunPositionGeographic (float * *x*, float * *y*, float * *z*) const

Retrieve the direction of the sun, relative to the center of the Earth in a geocentric coordinate system, where the z axis points from the center of the Earth to the North Pole, and x points toward the prime meridian.

11.2.3.67 static double SILVERLINING_API SilverLining::Atmosphere::GetUnitScale ( ) [inline, static]

Retrieves the scale factor for world units, as set by SetWorldUnits().

Defaults to 1.0.

11.2.3.68 void SILVERLINING_API SilverLining::Atmosphere::GetUpVector (double & *x*, double & *y*, double & *z*)

Returns the direction that SilverLining assumes is "up" as a unit vector.

11.2.3.69 bool SILVERLINING_API SilverLining::Atmosphere::GetViewport (int & *x*, int & *y*, int & *w*, int & *h*) const

Returns the viewport size and position last set via SetViewport().

11.2.3.70 void SILVERLINING_API SilverLining::Atmosphere::GetZenithColor (float * *r*, float * *g*, float * *b*) const

Returns the color of the sky at the zenith.

Parameters

r	A pointer to a float to receive the red channel of the zenith color, from 0 - 1.0

g	A pointer to a float to receive the red channel of the zenith color, from 0 - 1.0
b	A pointer to a float to receive the red channel of the zenith color, from 0 - 1.0

11.2.3.71 int SILVERLINING_API SilverLining::Atmosphere::Initialize (int *renderer*, const char * *resourceDirectoryPath*, bool *rightHanded*, void * *environment*)

Call this immediately after constructing your scene's Atmosphere and initializing your graphics subsystem (OpenGL, OpenGL32, DirectX9, DirectX10, DirectX11, or DirectX11.1).

This method will configure SilverLining to use OpenGL or DirectX, and in the case of DirectX, allows you to pass in a required pointer to your IDirect3DDevice9, ID3D10Device, or ID3D11Device object.

This method should be called from the same thread and context you intend to render the Atmosphere from.

If you are tying SilverLining directly into your own rendering engine, pass Atmosphere::CUSTOM_RENDERER as the renderer, provide your own implementation of the functions in SilverLiningDLLCommon.h, and link against the "static" versions of the SilverLining libraries.

The OPENGL32CORE renderer will only initialize successfully on graphics hardware and drivers that support the OpenGL 3.2 specification, with contexts created explicitly for OpenGL 3.2 or newer. It is compatible with forward-compatible OpenGL 3.2 core profile contexts that do not have backward-compatibilty enabled; no deprecated OpenGL functions are used in this renderer.

OpenGL users may pass 0 for the environment parameter. It is ignored for OpenGL.

You must also pass in a path to the Resources directory, which contains the art resources, data files, and shaders required for SilverLining to run. You may name and redistribute this directory however you wish, but SilverLining needs to know where it is and what it's called.

Parameters

renderer	Pass the enumerated constant `SilverLining::OPENGL` or `SilverLining::OPENGL32CORE` or `SilverLining::DIRECTX9` or `SilverLining::DIRECTX10` or `SilverLining::DIRECTX11` or `SilverLining::DIRECTX11_1` or `SilverLining::CUSTOM_RENDERER`
resourceDirectoryPath	A null-terminated string that specifies a path to the application's redistributed "Resources" directory, including the directory name itself. For example, "..\\Resources\\". If you pass NULL, the default path is ".\\Resources".

rightHanded	Pass true if you're using a right-handed coordinate system, false for a left-handed coordinate system. OpenGL typically uses right-handed; DirectX can use either.
environment	Only required for DIRECTX9, DIRECTX10, DIRECTX11, or DIRECTX11_1 renderers; a pointer to your IDirect3DDevice9, ID3D10Device, or ID3D11Device.

Returns

An error code from the Atmosphere::InitializeErrors enumeration, or E_NOERROR. See the troubleshooting section of the documentation for further guidance if you encounter an error.

11.2.3.72 int SILVERLINING_API SilverLining::Atmosphere::Initialize (int *renderer*, const char * *resourceDirectoryPath*, bool *rightHanded*, void * *environment*, const SL_VECTOR(unsigned int)& *userShaders*)

OpenGL-only variant of Atmosphere::Initialize() that takes in a list of user-compiled shader objects that will be linked into all subsequently linked shader program objects.

This allows you to inject your own shader functions into our shaders without copying the source.

11.2.3.73 bool SILVERLINING_API SilverLining::Atmosphere::IsInitialized ( ) const [inline]

Returns whether the Initialize() method has been successfully called on this Atmosphere yet, and it is safe to use for rendering tasks.

11.2.3.74 void SILVERLINING_API SilverLining::Atmosphere::OverrideCloudLighting (float *ambientR*, float *ambientG*, float *ambientB*, float *diffuseR*, float *diffuseG*, float *diffuseB*)

Use this to use specific colors for lighting the clouds, instead of the sun, moon, and sky colors.

See also the CloudLayer::OverrideCloudColor() method for overriding the diffuse light on a per cloud layer basis.

See also

ClearCloudLightingOverride()

Parameters

ambientR	The red component of the explicit color to use for ambient lighting of the clouds.
ambientG	The green component of the explicit color to use for ambient lighting of the clouds.
ambientB	The blue component of the explicit color to use for ambient lighting of the clouds.
diffuseR	The red component of the explicit color to use for diffuse (directional) lighting of the clouds.
diffuseG	The green component of the explicit color to use for diffuse (directional) lighting of the clouds.
diffuseB	The blue component of the explicit color to use for diffuse (directional) lighting of the clouds.

11.2.3.75 void SILVERLINING_API SilverLining::Atmosphere::ReloadConfigFile ( )

Reloads the SilverLining.config file in the resources directory.

This won't immediately affect existing clouds in the scene, but will affect new clouds created going forward. Mostly this is useful for advanced users who want to tweak the default config file settings without having to shut down and restart their application every time. Requires that Atmosphere::Initialize() was called previously.

11.2.3.76 bool SILVERLINING_API SilverLining::Atmosphere::ReloadCumulusTextures ( )

Reloads the textures used to represent cumulus cloud puffs.

When using CUMULUS_CONGESTUS_HI_RES for example, you might switch to a different texture set by first using Atmosphere::SetConfigOption("cloud-atlas-texture-name", [new texture name]) and then calling Atmosphere::ReloadCumulusTextures(). Returns true if the textures successfully loaded.

11.2.3.77 void SILVERLINING_API SilverLining::Atmosphere::ReloadShaders (const SL_VECTOR(unsigned int)& *userShaders*)

OpenGL only: recompile and link all shader programs on demand.

11.2.3.78 bool SILVERLINING_API SilverLining::Atmosphere::Serialize (std::ostream & *stream*)

Flatten the Atmosphere and everything in it to a stream - useful for capturing the state of everything to save to disk or for recreating the Atmosphere in response to a device reset in DX9.

11.2.3.79 void SILVERLINING_API SilverLining::Atmosphere::SetCameraMatrix (const double * *cameraMatrix*)

Call this at the beginning of each rendering loop, prior to calling DrawSky().

The matrix passed in should represent the transform for your camera, as a 4x4 matrix of doubles. Be sure to also call SetProjectionMatrix()

11.2.3.80 void SILVERLINING_API SilverLining::Atmosphere::SetConditions (const AtmosphericConditions & *conditions*)

Configures the simulated cloud, wind, time, and location.

The cloud, wind, time, and location settings are all contained inside an AtmosphericConditions object. Pass in an AtmosphericConditions class configured the way you like it, after first calling Initialize on the Atmosphere class, and before rendering your scene. Any previous conditions will be overwritten by this call.

Note, this passes in a const reference to your AtmosphericConditions class. It is copied internally, and you're free to dispose of your AtmosphericConditions object once you have passed it into SetConditions.

11.2.3.81 void SILVERLINING_API SilverLining::Atmosphere::SetConfigOption (const char * *key*, const char * *value*)

Sets a specific SilverLining.config value.

Will overwrite whatever was specified. Subject to the same caveats listed for ReloadConfigFile().

11.2.3.82 void SILVERLINING_API SilverLining::Atmosphere::SetDepthRange (float *nearDepth*, float *farDepth*)

Call this at the beginning of each rendering loop, prior to calling DrawSky(), indicating the range of depth buffer values in use.

Calling this is optional, but may result in avoiding a stall each frame. Normally, the near depth value is 0.0 and the far depth value is 1.0, unless you're doing something like implementing reversed floating point depth buffers.

Parameters

nearDepth	The z value that is mapped to the near clipping plane. (Usually 0)
farDepth	The z value that is mapped to the far clipping plane. (Usually 1)

11.2.3.83 void SILVERLINING_API SilverLining::Atmosphere::SetGamma (double *gamma*)

Sets the value used for gamma correction of the display.

Defaults to the sky-box-gamma setting. 1.8 works well. Higher values will yield lighter skies and natural light.

11.2.3.84 void SILVERLINING_API SilverLining::Atmosphere::SetHaze (float *hazeR*, float *hazeG*, float *hazeB*, double *hazeDepth*, double *hazeDensity*)

Causes the sky to blend toward a specified "haze color" toward the horizon.

Although it does simulate a layer of colored fog, it's most practical application to allow for exact blending against a fog color used for terrain, in order to obscure the horizon line. For applications that do not render terrain all the way to the horizon, this is a must. GetHorizonColor() may be used for an approximate match in the absence of an artificial layer of haze; it is more physically accurate.

The haze color passed in is not lit; you must pre-multiply the color yourself. The skybox will blend toward the exact color passed in at the horizon, night or day. You could abuse this to create a glow effect at the horizon from city lights, for example.

By default, hazeDepth is set to 0, thereby disabling the haze effects. If the viewpoint is within a cloud, the fog effects from the cloud will drawn in the sky in lieu of haze.

Parameters

hazeR	The red component of the color to blend toward at the horizon.
hazeG	The red component of the color to blend toward at the horizon.
hazeB	The red component of the color to blend toward at the horizon.
hazeDepth	The simulated height of the haze volume at ground level, in world units.
hazeDensity	The fog density parameter of the exponential fog equation.

11.2.3.85 void SILVERLINING_API SilverLining::Atmosphere::SetInfraRedMode (bool *bInfraRed*) [inline]

Sets infrared sensor simulator mode.

Just renders everything as black except the sun.

11.2.3.86 void SILVERLINING_API SilverLining::Atmosphere::SetMoonAlpha (double *alpha*)

Sets a transparency value for the moon billboard, useful for fading the moon out with haze.

Parameters

alpha	- The transparency of the moon billboard; 0 = transparent, 1 = opaque.

11.2.3.87 void SILVERLINING_API SilverLining::Atmosphere::SetOutputScale (float *scale*) [inline]

Scales all final color values by some factor; useful if you need to match SilverLining to some other tone mapping scheme or just want a quick way to darken or lighten everything.

Parameters

scale	The scaling factor for the final output in all fragment shaders.

11.2.3.88 void SILVERLINING_API SilverLining::Atmosphere::SetProjectionMatrix (const double * *projectionMatrix*)

Call this at the beginning of each rendering loop, prior to calling DrawSky().

The matrix passed in should represent the projection matrix for your scene, as a 4x4 matrix of doubles. Be sure to also call SetCameraMatrix()

11.2.3.89 void SILVERLINING_API SilverLining::Atmosphere::SetRandomNumberGenerator (const RandomNumberGenerator * *rng*)

If you wish to override SilverLining's default random number generator, which is based on the standard library's rand() function, you may do so by extending the RandomNumberGenerator base class, instantiating your extended class, and passing in a pointer to your class here.

You are responsible for deleting the RandomNumberGenerator you passed in at shutdown.

11.2.3.90 static void SILVERLINING_API SilverLining::Atmosphere::SetResourceLoader (ResourceLoader * *loader*) [inline, static]

Overrides the default FILE* - based resource loader with a customer-supplied resource loader object.

See the ResourceLoader documentation for more information. This allows you to include SilverLining's textures, models, and shaders within your own resource management scheme. This should be called after initializing your first Atmosphere object,

but before calling Atmosphere::Initialize(). You are responsible for deleting the ResourceLoader you passed in at shutdown.

11.2.3.91 void SILVERLINING_API SilverLining::Atmosphere::SetRightVector (double *x*, double *y*, double *z*)

Sets the assumption of what direction is "right".

If the vector (x, y, z) is not a unit vector, it is normalized before being stored. Cannot be called prior to Atmosphere::Initialize(). Must be called in conjunction with SetUpVector().

Be sure to call this prior to positioning any clouds.

See also

SetUpVector()
GetRightVector()

11.2.3.92 void SILVERLINING_API SilverLining::Atmosphere::SetSkyModel (SkyModel *skyModel*)

Sets the physical model used for simulating sky colors.

The "Preetham model" is simple and fast, but has some inaccuracies near the horizon. The newer "Hosek-Wilkie" model extends the Preetham model for more accurate sky colors, especially at very high and low solar angles. If "Hosek-Wilkie" is selected, it's actually only used for daytime lighting form the sun - twilight and moonlit conditions still use the Preetham model, since Hosek-Wilkie can only simulate sky colors from sunlight at positive solar angles.

Parameters

skyModel	One of PREETHAM or HOSEK_WILKIE, as described above.

11.2.3.93 void SILVERLINING_API SilverLining::Atmosphere::SetSunAlpha (double *alpha*)

Sets a transparency value for the sun billboard, useful for fading the sun out with haze.

Parameters

alpha	- The transparency of the sun billboard; 0 = transparent, 1 = opaque.

11.2.3.94 void SILVERLINING_API SilverLining::Atmosphere::SetUpVector (double *x*, double *y*, double *z*)

Sets the assumption of what direction is "up".

If the vector (x, y, z) is not a unit vector, it is normalized before being stored. Cannot be called prior to Atmosphere::Initialize(). Must be called in conjunction with SetRightVector().

Be sure to call this prior to positioning any clouds.

In a geocentric / ECEF coordinate system, this should be the normalized camera position vector pointing from the center of the Earth, not the local normal vector of the Earth's ellipsoid. There is a subtle difference between the two that can lead to cloud positions being a bit off.

See also

SetRightVector()
GetUpVector()

11.2.3.95 void SILVERLINING_API SilverLining::Atmosphere::SetViewport (int *x*, int *y*, int *w*, int *h*)

Call this at the beginning of each rendering loop, prior to calling DrawSky().

Parameters

x	The x position of the viewport origin.
y	The y position of the viewport origin.
w	The width of the viewport.
h	The height of the viewport.

11.2.3.96 static void SILVERLINING_API SilverLining::Atmosphere::SetWorldUnits (double *meters*) [inline, static]

By default, SilverLining assumes that one world unit is equal to one meter - that is, all of the dimensions for cloud layers, altitudes, etc.

are in meters. If you are using a different scale, you may specify the size of one unit in meters here. This should be set prior to initializing the Atmosphere object.

11.2.3.97 SilverLining::Atmosphere::SL_VECTOR (unsigned *int*)

OpenGL only: retrieve a list of OpenGL shader program objects for all currently active planar cloud shaders.

This may be used for attaching your own shader uniform parameters.

11.2.3.98 SilverLining::Atmosphere::SL_VECTOR (ObjectHandle) const

If you called DrawObjects() with the drawClouds parameter set to false, no clouds will actually be drawn.

It's then your responsibility to obtain a list of cloud objects to draw, and draw them yourself. GetObjects() provides you with this list. This list will be empty until DrawObjects() has been called for the current frame. If you call DrawObjects() with the drawClouds parameter set to true, this list will also be empty, since the objects have already been drawn.

See also

DrawObjects()
GetObjectDistance()
DrawObject()

11.2.3.99 bool SILVERLINING_API SilverLining::Atmosphere::Unserialize (std::istream & *stream*)

Restore the Atmosphere from a stream generated from Atmosphere::Serialize.

The Atmosphere must be constructed and initialized before calling Unserialize().

11.2.3.100 void SILVERLINING_API SilverLining::Atmosphere::UpdateEphemeris ( )

Forces the ephemeris model to recalculate the position of the sun, moon, and stars for the currently simulated time and location.

This happens automatically when DrawSky() or UpdateSkyAndClouds() is called; this is provided only if you need to compute the astronomical positions in response to a change in the simulated time and location prior to calling DrawSky() or UpdateSkyAndClouds().

11.2.3.101 void SILVERLINING_API SilverLining::Atmosphere::UpdateSkyAndClouds ( )

Explicitly updates the ephemeris model and cloud shapes and positions.

Calling this method is optional; if it is not called prior to Atmosphere::DrawSky(), it will be called automatically. But, once you start calling it, it must be called every frame. This exposes the ability to perform updates in a separate pass or thread from the actual drawing of the sky and clouds. However, you must ensure that a valid and correct GL or device context is active prior to calling this. For example, this is not the case with OpenSceneGraph's update traversal - so don't use this method with OSG.

The documentation for this class was generated from the following file:

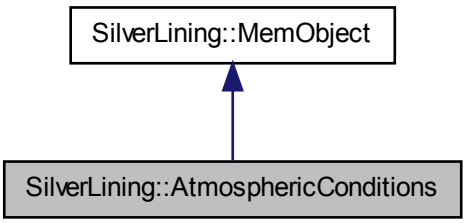
- C:/sl/Public Headers/Atmosphere.h

11.3 SilverLining::AtmosphericConditions Class Reference

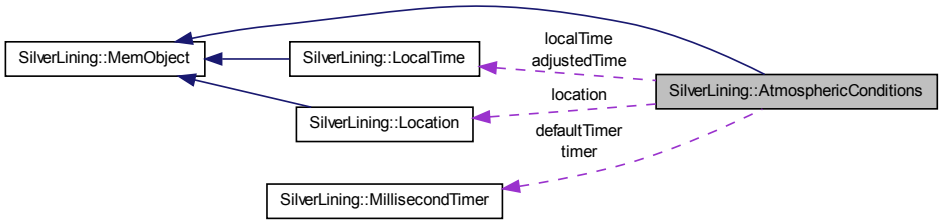
A class to set the current simulation conditions for the sky.

`#include <AtmosphericConditions.h>`

Inheritance diagram for SilverLining::AtmosphericConditions:



Collaboration diagram for SilverLining::AtmosphericConditions:



Public Member Functions

- AtmosphericConditions ()

  Default constructor.

- virtual ~AtmosphericConditions ()

  Destructor.

- void SILVERLINING_API SetLocation (const Location &location)

  Sets the simulated geographic location.

- const Location &SILVERLINING_API GetLocation () const

  Gets the current simulated location.

- void SILVERLINING_API SetTime (const LocalTime &time)

  Sets the simulated time of day in local time.

- const LocalTime &SILVERLINING_API GetTime ()

  Retrieves the currently simulated time of day.

- void SILVERLINING_API SetPresetConditions (ConditionPresets preset, Atmosphere &atm)

  A quick way to set up typical weather conditions.

- int SILVERLINING_API SetWind (const WindVolume &windVolume)

  Sets a volume of wind for moving clouds.

- bool SILVERLINING_API RemoveWindVolume (int windVolumeHandle)

Removes a specific wind volume from the simulation.

- void SILVERLINING_API ClearWindVolumes ()

  Clears all wind volumes previously set via SetWind.

- void SILVERLINING_API GetWind (double &velocity, double &heading, double altitude) const

  Retrieves the simulated wind at a given altitude.

- void SILVERLINING_API ApplyWind (double dt)

  Apply wind to cloud layers for given time step.

- const SL_MAP (int, WindVolume)&SILVERLINING_API GetWindVolumes() const

  Get a vector of all wind volumes being simulated.

- int SILVERLINING_API AddCloudLayer (CloudLayer *layer)

  Adds a cloud layer to the simulated atmospheric conditions.

- bool SILVERLINING_API RemoveCloudLayer (int layerHandle)

  Removes a previously added cloud deck from the scene.

- void SILVERLINING_API RemoveAllCloudLayers ()

  Removes all cloud layers from the scene.

- bool SILVERLINING_API GetCloudLayer (int layerHandle, CloudLayer **layer)

  Obtain a pointer to a CloudLayer object.

- SL_MAP (int, CloudLayer *)&SILVERLINING_API GetCloudLayers()

  Obtain the stl map of CloudLayer objects.

- void SILVERLINING_API SetTurbidity (double pTurbidity)

  Sets the "turbidity" of the simulated atmosphere.

- double SILVERLINING_API GetTurbidity () const

  Get the simulated atmospheric turbidity.

- void SILVERLINING_API SetVisibility (double range)

  Sets the simulated visibility in meters; this will affect the appearance of clouds in the distance.

- double SILVERLINING_API GetVisibility () const

Retrieves the currently simulated visibility, in meters.

- void SILVERLINING_API SetFog (double density, double r, double g, double b)

  Explicitly sets exponential fog color and density, if you need to match an existing scene.

- void SILVERLINING_API ClearFog ()

  Clears explicitly set fog from SetFog(), and reverts the fog to simulating the visibility specified in SetVisibility() instead.

- void SILVERLINING_API GetFog (bool &fogIsSet, double &density, double &r, double &g, double &b)

  Retrieves the explicit fog (if any) set by SetFog().

- void SILVERLINING_API SetLightPollution (double Wm2)

  Sets the simulated amount of nighttime light pollution, in watts per square meter.

- double SILVERLINING_API GetLightPollution () const

  Retrieves the currently simulated light pollution, in watts per square meter.

- void SILVERLINING_API SetPrecipitation (int precipitationType, double precipitationRate, double nearClip=-1, double farClip=-1, bool bUseDepthBuffer=false)

  Simulates global precipitation of a specified type.

- void SILVERLINING_API SetPrecipitationWind (double windX, double windZ)

  Sets a precipitation specific wind vector which is additive to any global wind set at the AtmosphericConditions level.

- void SILVERLINING_API GetPrecipitationWind (double &windX, double &windZ) const

  Retrieves the precipitation specific wind vector.

- void SILVERLINING_API EnableTimePassage (bool enabled, long relightFrequencyMS)

  By default, SilverLining will "freeze" time at the time specified by AtmosphericConditions::SetTime().

- void SILVERLINING_API SetMillisecondTimer (const MillisecondTimer *timer)

  By default, SilverLining will simulate the motion of clouds in the wind and the motion of astronomical objects (when EnableTimePassage() is active) by calling the system's millisecond timer.

- const MillisecondTimer *SILVERLINING_API GetMillisecondTimer () const

  Retrieves the MillisecondTimer previously set by SetMillisecondTimer(), or the default timer if SetMillisecondTimer() is unset or set to NULL.

- bool SILVERLINING_API Serialize (std::ostream &stream)

  Flattens this object and everything in it to a stream buffer.

- bool SILVERLINING_API Unserialize (const Atmosphere *atm, std::istream &stream)

  Restores this object from the stream created using Serialize()

11.3.1 Detailed Description

A class to set the current simulation conditions for the sky. This class allows you to specify the simulated time, location, wind, visibility, haze, and cloud cover for your scene. You do so by instantiating your own AtmosphericConditions object, configuring it by calling its methods, and then passing it into your Atmosphere object via Atmosphere::SetConditions().

To make changes to the AtmosphericConditions after it has been passed into Atmosphere::SetConditions(), call Atmosphere::GetConditions() and call methods on the AtmosphericConditions reference that it returns. Atmosphere::SetConditions() copies your AtmosphericConditions into the Atmosphere class, so changes made to the AtmosphericConditions used to initialize the Atmosphere will have no effect.

11.3.2 Constructor & Destructor Documentation

11.3.2.1 SilverLining::AtmosphericConditions::AtmosphericConditions ( )

Default constructor.

Creates an AtmosphericConditions object with default settings.

11.3.2.2 virtual SilverLining::AtmosphericConditions::~AtmosphericConditions ( ) [virtual]

Destructor.

Destroys all members, but does not destroy the CloudLayer objects passed into AddCloudLayer().

11.3.3 Member Function Documentation

11.3.3.1 int SILVERLINING_API SilverLining::AtmosphericConditions::AddCloudLayer (CloudLayer * *layer*)

Adds a cloud layer to the simulated atmospheric conditions.

Cloud layers are created via the CloudLayerFactory::Create() method, and won't become part of your scene until they are added to a AtmosphericConditions class that is then passed into Atmosphere::SetConditions(). The CloudLayer object will be destroyed along with this AtmosphericConditions class; there's no need to keep this object around for cleanup.

Parameters

layer	A CloudLayer created via CloudLayerFactory::Create(). This CloudLayer should be configured and seeded (see CloudLayer::SeedClouds()) prior to being passed in.

Returns

An integer handle to the cloud layer that may be used in calls to RemoveCloudLayer() and GetCloudLayer().

11.3.3.2 void SILVERLINING_API SilverLining::AtmosphericConditions::ApplyWind (double *dt*)

Apply wind to cloud layers for given time step.

This is called by the Atmosphere class; you do not need to call this directly!

Parameters

dt	Length of time to simulate wind over, in seconds.

11.3.3.3 void SILVERLINING_API SilverLining::AtmosphericConditions::ClearFog ( )

Clears explicitly set fog from SetFog(), and reverts the fog to simulating the visibility specified in SetVisibility() instead.

11.3.3.4 void SILVERLINING_API SilverLining::AtmosphericConditions::ClearWindVolumes ( )

Clears all wind volumes previously set via SetWind.

11.3.3.5 void SILVERLINING_API SilverLining::AtmosphericConditions::EnableTimePassage (bool *enabled,* long *relightFrequencyMS*)

By default, SilverLining will "freeze" time at the time specified by AtmosphericConditions::SetTime().

If you want to simulate the passage of time, call EnableTimePassage with the enabled parameter set to true.

Relighting the clouds is a relatively expensive operation, so for real time applications you probably won't want to relight the clouds every frame. The relightFrequencyMS parameter allows you to specify the interval, in milliseconds, between cloud relighting passes. The sky will continue to update in real time, along with the position of the sun, moon, and stars, between these intervals. If you have specified your own MillisecondTimer with SetMillisecondTimer(), the interval will be computed based on its concept of time.

If the enabled parameter is false (the default,) or the relightFrequencyMS parameter is set to -1, cloud relighting will only happen in response to calls to AtmosphericConditions::SetTime().

Clouds will move with respect to the simulated wind irregardless of calling this method.

Parameters

enabled	True if you want to simulate the passage of time between calls to SetTime(); false if the simulated time should remain static.
relightFrequencyMS	The interval, in milliseconds, between cloud relighting passes. Set to -1 to prevent cloud relighting outside of calls to SetTime().

11.3.3.6 bool SILVERLINING_API SilverLining::AtmosphericConditions::GetCloudLayer (int *layerHandle,* **CloudLayer** ** *layer*)

Obtain a pointer to a CloudLayer object.

Given a cloud layer handle, obtain its underlying CloudLayer object.

Parameters

layerHandle	The integer handle to a cloud layer previously returned by AddCloudLayer().
layer	A pointer to a pointer to receive the CloudLayer object pointer requested.

Returns

false if the layerHandle does not specify a CloudLayer being managed by this class.

11.3.3.7 void SILVERLINING_API SilverLining::AtmosphericConditions::GetFog (bool & *fogIsSet,* double & *density,* double & *r,* double & *g,* double & *b*)

Retrieves the explicit fog (if any) set by SetFog().

Parameters

fogIsSet	Returns true if SetFog has been called and ClearFog has not, meaning we are using fog with an explicitly defined application setting instead of simulating visibility.
density	The exponential fog density term set in SetFog()
r	The red component of the fog color specified in SetFog(). Ranges from 0-1.
g	The green component of the fog color specified in SetFog(). Ranges from 0-1.
b	The blue component of the fog color specified in SetFog(). Ranges from 0-1.

11.3.3.8 double SILVERLINING_API SilverLining::AtmosphericConditions::GetLightPollution ( ) const [inline]

Retrieves the currently simulated light pollution, in watts per square meter.

11.3.3.9 const Location& SILVERLINING_API SilverLining::AtmosphericConditions::GetLocation ( ) const

Gets the current simulated location.

Returns

A const reference to a Location object that may be queried for the current simulated geographic position.

11.3.3.10 const MillisecondTimer* SILVERLINING_API SilverLining::AtmosphericConditions::GetMillisecondTimer ( ) const [inline]

Retrieves the MillisecondTimer previously set by SetMillisecondTimer(), or the default timer if SetMillisecondTimer() is unset or set to NULL.

11.3.3.11 void SILVERLINING_API SilverLining::AtmosphericConditions::GetPrecipitationWind (double & *windX,* double & *windZ*) const [inline]

Retrieves the precipitation specific wind vector.

See also

SetWind()

11.3.3.12 const LocalTime& SILVERLINING_API SilverLining::AtmosphericConditions::GetTime ( )

Retrieves the currently simulated time of day.

Use this to obtain the local time, time zone, and daylight savings time observance being simulated.

This will also reflect the passage of time simulated by activating EnableTimePassage(). As such, it may differ from what was originally passed into SetTime() if time passage is activated.

Returns

A const reference to a LocalTime object, which may be queried for the local time, time zone, and DST information being simulated.

11.3.3.13 double SILVERLINING_API SilverLining::AtmosphericConditions::GetTurbidity ( ) const [inline]

Get the simulated atmospheric turbidity.

See SetTurbidity() for an explanation of turbidity values.

11.3.3.14 double SILVERLINING_API SilverLining::AtmosphericConditions::GetVisibility ( ) const [inline]

Retrieves the currently simulated visibility, in meters.

11.3.3.15 void SILVERLINING_API SilverLining::AtmosphericConditions::GetWind (double & *velocity*, double & *heading*, double *altitude*) const

Retrieves the simulated wind at a given altitude.

Parameters

velocity	Receives the simulated wind velocity in meters per second
heading	Receives the simulated wind direction in degrees from North
altitude	The altitude, in meters, for which you want wind information.

11.3.3.16 void SILVERLINING_API SilverLining::AtmosphericConditions::RemoveAllCloudLayers ( )

Removes all cloud layers from the scene.

All CloudLayer objects currently contained by this class will be removed and deleted.

11.3.3.17 bool SILVERLINING_API SilverLining::AtmosphericConditions::RemoveCloudLayer (int *layerHandle*)

Removes a previously added cloud deck from the scene.

The specified CloudLayer handle, if found, will be removed from the simulation and its CloudLayer object deleted.

Parameters

layerHandle	The integer handle previously returned from AddCloudLayer().

Returns

true if the cloud deck was removed, false if the handle specified was not found.

11.3.3.18 bool SILVERLINING_API SilverLining::AtmosphericConditions::RemoveWindVolume (int *windVolumeHandle*)

Removes a specific wind volume from the simulation.

Parameters

windVolumeHandle	The handle returned from AtmosphericConditions::SetWind().

11.3.3.19 bool SILVERLINING_API SilverLining::AtmosphericConditions::Serialize (std::ostream & *stream*)

Flattens this object and everything in it to a stream buffer.

11.3.3.20 void SILVERLINING_API SilverLining::AtmosphericConditions::SetFog (double *density*, double *r*, double *g*, double *b*)

Explicitly sets exponential fog color and density, if you need to match an existing scene.

If set, this will override the visibility effects from SetVisibility(). Color components are specified in the range 0-1. Density should be 1.0 / simulated visibility. This affects the appearance of the clouds, but not the sky. This is intended for real, thick fog, as opposed to SetVisibility(), which simulates extinction from atmospheric scattering. When using this, you'll usually want to call DrawSky() with false in the first parameter to suppress drawing the sky box, and just clear your back buffer to the fog color instead. If you do draw the sky, you may want to consider disabling the drawing of the stars, sun, and moon when calling DrawSky() if thick fog is present.

11.3.3.21 void SILVERLINING_API SilverLining::AtmosphericConditions::SetLightPollution (double *Wm2*) [inline]

Sets the simulated amount of nighttime light pollution, in watts per square meter.

Defaults to zero. A reasonable value would be in the order of 0.01

11.3.3.22 void SILVERLINING_API SilverLining::AtmosphericConditions::SetLocation (const Location & *location*)

Sets the simulated geographic location.

This affects the sun and moon positions relative to the horizon, and thereby also affects the lighting of the scene. Be sure that the location specified is consistent with the time zone specified in the LocalTime passed to SetTime(), or you will experience confusing results.

Parameters

location	A reference to a Location object that embodies the simulated latitude and longitude.

11.3.3.23 void SILVERLINING_API SilverLining::AtmosphericConditions::SetMillisecondTimer (const MillisecondTimer * *timer*)

By default, SilverLining will simulate the motion of clouds in the wind and the motion of astronomical objects (when EnableTimePassage() is active) by calling the system's millisecond timer.

If you want to accelerate, slow, or reverse the passage of time, you may instead provide your own MillisecondTimer implementation, and pass it in here. See the documentation for the MillisecondTimer class for more details.

Parameters

timer	The MillisecondTimer object to use for moving clouds and astronomical objects over time, relative to the LocalTime passed into SetTime(). Pass NULL to restore the default timer.

11.3.3.3.24 void SILVERLINING_API SilverLining::AtmosphericConditions::SetPrecipitation (int *precipitationType*, double *precipitationRate*, double *nearClip* = -1, double *farClip* = -1, bool *bUseDepthBuffer* = false)

Simulates global precipitation of a specified type.

Precipitation effects will display if the type is set to something other than NONE.

Note, you may also set precipitation effects associated with a CloudLayer that only render when the camera is underneath a rain cloud with the similar CloudLayer::SetPrecipitation() method.

For mixed precipitation, you may call SetPrecipitation multiple times with different precipitation types. To clear all precipitation, call SetPrecipitation with a type of CloudLayer::NONE. If you call this method multiple times for the same precipitation type, the intensity specified will overwrite the intensity previously specified for that type.

Parameters

precipitationType	The type of precipitation to simulate under this cloud layer - CloudLayer::NONE, CloudLayer::RAIN, CloudLayer::WET_SNOW, CloudLayer::DRY_SNOW, or CloudLayer::SLEET.
precipitationRate	The simulated rate of precipitation, in millimeters per hour. Reasonable ranges might be between 1 for light rain or 20 for heavier rain. This value will be clamped to the value specified by rain-max-intensity, snow-max-intensity, or sleet-max-intensity in resources/SilverLining.config, which is 30 by default.
nearClip	How close to the camera the closest precipitation particles will be rendered. The near clipping plane will be adjusted to include this distance while the precipitation is being renderered. Values less than zero will result in the default value being applied.
farClip	The farthest distance from the camera that precipitation particles will be rendered. Since there is an upper bound on the number of particles rendered per frame, changing the difference between nearClip and farClip may result in changes to the intensity of the precipitation. By default this difference is 7.5 world units. Values less than zero will result in the default value being applied.
bUseDepthBuffer	Set this to true if you want to enable depth buffer testing of precipitation particles against your scene.

11.3.3.25 void SILVERLINING_API SilverLining::AtmosphericConditions::SetPrecipitationWind (double *windX*, double *windZ*) [inline]

Sets a precipitation specific wind vector which is additive to any global wind set at the AtmosphericConditions level.

The vector itself specifies the wind direction and the length of the vector provides the wind velocity.

Parameters

windX	The east vector of the precipitation specific wind, in meters/s.
windZ	The south vector of the precipitation specific wind, in meters/s.

See also

GetWind()

11.3.3.26 void SILVERLINING_API SilverLining::AtmosphericConditions::SetPresetConditions (ConditionPresets *preset*, Atmosphere & *atm*)

A quick way to set up typical weather conditions.

This method will create "infinite" cloud layers that remain centered at the camera, so there's no need to worry about positioning them. Typical, realistic values for cloud altitudes will be used. Finer control is available by setting up your own CloudLayer objects and passing them into AtmosphericConditions::AddCloudLayer().

To simulate storms, set up MOSTLY_CLOUDY conditions, and then position a few Cumulonimbus clouds near the camera in addition.

Any existing cloud layers in the scene will be removed. You'll still need to set up the simulated time, location, visibility, and any precipitation and wind you might want.

Parameters

preset	The cloud condition preset you wish to use; one of AtmosphericConditions::FAIR, PARTLY_CLOUDY, MOSTLY_CLOUDY, or OVERCAST.
atm	The Atmosphere object to associate the clouds with.

11.3.3.27 void SILVERLINING_API SilverLining::AtmosphericConditions::SetTime (const LocalTime & *time*)

Sets the simulated time of day in local time.

You must specify if daylight savings time is currently being observed or not. Be sure

that the time zone specified is consistent with the simulated Location passed to SetLocation(), or you will experience confusing results.

Parameters

time	A reference to a LocalTime object that embodies the simulated local time, time zone, and daylight savings time observation.

11.3.3.28 void SILVERLINING_API SilverLining::AtmosphericConditions::SetTurbidity (double *pTurbidity*) [inline]

Sets the "turbidity" of the simulated atmosphere.

You can think of this as a measure of "haziness." Some guidelines for setting this value:

2 = very clear, range 50 km 3 = clear, range 15 km 7 = light haze, range 8 km

Setting turbidity is not an appropriate way to simulate fog; see Atmosphere::SetHaze() for one method, or for dense fog simply clear the backbuffer to the fog color and pass false to Atmosphere::DrawSky() to suppress the sky rendering altogether.

Turbidity just lets you simulate the number of particles in the air. In practice it will vary the color of the sky from a pure light blue to a hazy, darker, yellowish color. Turbidity values below 1.8 or above 8.0 lead to unpredictable results and are clamped.

Parameters

pTurbidity	The ratio of scattering due to haze to scattering due to molecules.

11.3.3.29 void SILVERLINING_API SilverLining::AtmosphericConditions::SetVisibility (double *range*) [inline]

Sets the simulated visibility in meters; this will affect the appearance of clouds in the distance.

Defaults to 30km. This is intended only for light haze, and serves only to blend clouds into the sky in the distance. It does not fog the sky itself. For thicker fog, see AtmosphericConditions::SetFog() and Atmosphere::SetHaze().

11.3.3.30 int SILVERLINING_API SilverLining::AtmosphericConditions::SetWind (const WindVolume & *windVolume*)

Sets a volume of wind for moving clouds.

You may call this multiple times to define different wind velocities and directions at different altitudes. Wind also has some influence on cloud formation and the rendering

of virga effects.

Parameters

windVolume	A WindVolume to add to the simulation.

Returns

A handle to the wind volume

11.3.3.31 SilverLining::AtmosphericConditions::SL_MAP (int , CloudLayer *) [inline]

Obtain the stl map of CloudLayer objects.

An accessor for a reference to the STL map of CloudLayer objects managed by this AtmosphericConditions class. It is not const, and so it may be directly manipulated. Use with care.

11.3.3.32 const SilverLining::AtmosphericConditions::SL_MAP (int , WindVolume) const [inline]

Get a vector of all wind volumes being simulated.

Returns

A const reference to the STL vector of wind volumes currently being simulated.

The documentation for this class was generated from the following file:

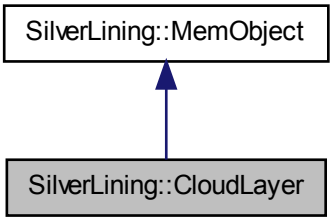
- C:/sl/Public Headers/AtmosphericConditions.h

11.4 SilverLining::CloudLayer Class Reference

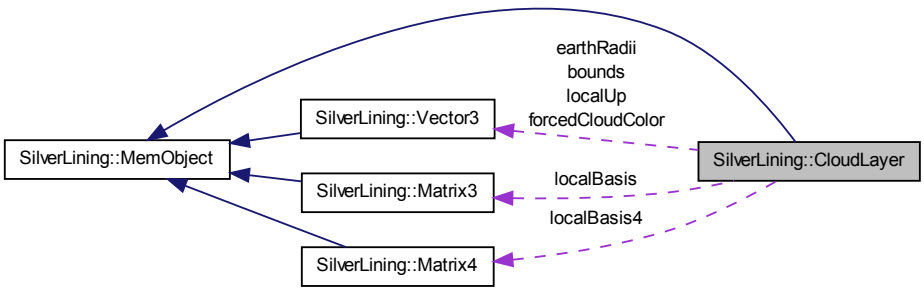
A collection of clouds of uniform classification organized into a "deck" of similar altitudes.

`#include <CloudLayer.h>`

Inheritance diagram for SilverLining::CloudLayer:



Collaboration diagram for SilverLining::CloudLayer:



Public Member Functions

- CloudLayer ()

  Default constructor.

- virtual ~CloudLayer ()

  Destructor.

- void ClearClouds ()

Removes all clouds from this cloud layer.

- CloudTypes SILVERLINING_API GetType () const

  Returns the cloud type of this layer.

- void SILVERLINING_API SetType (CloudTypes type)

  Sets the type of this cloud layer.

- bool SILVERLINING_API Intersect (const Vector3 &Origin, const Vector3 &Direction, double &range) const

  Test if ray intersects this cloud layer.

- void SILVERLINING_API SetEnabled (bool enabled, unsigned long fadeTimeMS=0)

  Enable rendering of this cloud layer.

- bool SILVERLINING_API GetEnabled () const

  Accesses if the CloudLayer is currently enabled for rendering.

- bool SILVERLINING_API IsRenderable () const

  Returns whether this cloud layer needs to be rendered at all; if it is disabled via SetEnabled() and any fading time has expired, this will return false.

- virtual void SILVERLINING_API SetBaseWidth (double meters)

  Sets the size of the CloudLayer along the X axis.

- double SILVERLINING_API GetBaseWidth () const

  Retrieves the dimension of the CloudLayer along the X axis.

- virtual void SILVERLINING_API SetBaseLength (double meters)

  Sets the size of the CloudLayer along the Z axis.

- double SILVERLINING_API GetBaseLength () const

  Retrieves the dimension of the CloudLayer along the Z axis.

- virtual void SILVERLINING_API SetBaseAltitude (double meters, bool updateCloudPositions=true)

  Sets the altitude above sea level of the bottom of the CloudLayer.

- double SILVERLINING_API GetBaseAltitude () const

  Retrieves the altitude above sea level of the CloudLayer botton in meters.

- virtual void SILVERLINING_API SetThickness (double meters)

Sets the thickness of the CloudLayer in meters.

- double SILVERLINING_API GetThickness () const

  Retrieves the thickness of the CloudLayer in meters.

- virtual void SILVERLINING_API SetDensity (double coveragePercentage)

  Sets the percentage of the sky that should be covered by this CloudLayer, from 0 - 1.0.

- double SILVERLINING_API GetDensity () const

  Retrieves the cloud coverage set for this CloudLayer.

- void SILVERLINING_API SetLayerPosition (double eastCoord, double southCoord)

  Specifies the location of the center of the CloudLayer.

- void SILVERLINING_API GetLayerPosition (double &east, double &south) const

  Retrieves the east and south coordinates of the CloudLayer's center, in meters.

- virtual void SILVERLINING_API SetIsInfinite (bool inf)

  Selects whether this cloud layer is "infinite" - that is, it will always surround the camera when feasible, and never gets blown away by wind.

- bool SILVERLINING_API GetIsInfinite () const

  Returns whether this cloud layer is "infinite" and attempts to remain fixed relative to the camera position as best it can.

- void SILVERLINING_API SetWind (double windX, double windZ)

  Sets a cloud layer specific wind vector which is additive to any global wind set at the AtmosphericConditions level.

- void SILVERLINING_API GetWind (double &windX, double &windZ) const

  Retrieves the cloud layer specific wind vector.

- virtual void SILVERLINING_API SetCloudAnimationEffects (double voxelSpinRate, bool enableGrowth, int initialGrowthIterations=0, int timeStepInterval=0)

  Controls cloud animation effects over time.

- void SILVERLINING_API SetCloudWrapping (bool wrap)

  Infinite cloud layers will always wrap the clouds around the camera position, but you can also enable this behavior relative to the position of a non-infinite cloud layer.

- bool SILVERLINING_API GetCloudWrapping () const

Returns whether non-infinite cloud layer types will wrap wind-blown clouds to remain within the original bounds of the layer, or if wind will move the entire layer indefinitely.

- void SILVERLINING_API SetFadeTowardEdges (bool fade)

  For infinite cloud layers or cloud layers with "wrapping" enabled this option allows you to automatically fade clouds out as they approach the boundary of the cloud layer where they will pop out and reappear on the opposite side of the layer.

- bool SILVERLINING_API GetFadeTowardEdges () const

  Retrieves whether an infinite or wrapping cloud layer has fade effects enabled as the clouds approach the boundaries of the layer.

- void SILVERLINING_API SetPrecipitation (int type, double intensity, double nearClip=-1.0, double farClip=-1.0, bool useDepthBuffer=false)

  Simulates precipitation under this cloud layer of a specified type.

- const SL_MAP (int, double)&SILVERLINING_API GetPrecipitation() const

  Retrieves a map of precipitation types to intensities in millimeters per hour that are associated with this cloud layer.

- virtual bool SILVERLINING_API HasPrecipitationAtPosition (double x, double y, double z) const

  Returns whether a precipitation type other than NONE will be simulated at the given camera position.

- virtual bool SILVERLINING_API IsInsideCloud (double x, double y, double z) const

  Returns whether the given location is inside the bounding box of any of the clouds within this cloud layer.

- void SILVERLINING_API SetAlpha (double pAlpha)

  Sets translucency for the cloud layer; 0 = transparent, 1.0 = normal opacity.

- double SILVERLINING_API GetAlpha () const

  Retrieves explicit translucency set by SetAlpha().

- virtual bool SILVERLINING_API SeedClouds (const Atmosphere &atm)=0

  Generates clouds within the cloud layer.

- virtual bool SILVERLINING_API AddCloudAt (const Atmosphere &atm, const Vector3 &relativePosition, const Vector3 &dimensions)

  Adds an individual cloud to an already-seeded cloud layer.

- virtual bool SILVERLINING_API SupportsAddCloudAt () const

  Indicates whether this cloud layer supports the CloudLayer::AddCloudAt() method for adding user-positioned clouds to the procedural clouds.

- void SILVERLINING_API AddLightningListener (LightningListener *listener)

  Adds a LightningListener - derived class to the list of LightningListeners to receive notifications of lightning strikes created by this CloudLayer.

- void SILVERLINING_API ClearLightningListeners ()

  Clears the list of LightningListener objects that receive notifications of lightning strikes from this CloudLayer.

- virtual bool SILVERLINING_API ForceLightning (bool value=true)

  Forces a lightning strike starting with this frame on cloud layers that support lightning.

- virtual void SILVERLINING_API SetLightningDischargeMode (LightningDischargeMode mode)

  Sets the lightning mode.

- virtual LightningDischargeMode SILVERLINING_API GetLightningDischargeMode ()

  Returns the current lightning discharge mode for this cloud layer, or MAX_DISCHARGE_MODE on layer that don't support lightning.

- virtual bool SILVERLINING_API Save (const char *filePath)

  Saves this cloud deck's clouds to a file for fast restoring later.

- virtual bool SILVERLINING_API Restore (const Atmosphere &atm, const char *filePath)

  Restores a cloud deck from a file previously generated with the Save() method.

- virtual bool SILVERLINING_API ExportToVRML (const char *dirPath)

  Exports each individual cloud into a numbered .WRL file within the specified directory path.

- virtual bool SILVERLINING_API Serialize (std::ostream &stream)

  Flattens this object and everything in it to a stream buffer.

- virtual bool SILVERLINING_API Unserialize (const Atmosphere &atm, std::istream &stream)

  Restores this object from the stream created using Serialize()

- const SL_VECTOR (Cloud ∗)&SILVERLINING_API GetClouds() const

  Retrieve the vector of underlying cloud objects (useful only to licensed users with the full source and private headers.)

- virtual double SILVERLINING_API GetMaxHeight () const

  Retrieves the height of the bounding box enclosing this cloud layer.

- virtual void SILVERLINING_API SetDensityMultiplier (double multiplier)

  This allows you to reduce the coverage of cloud layers that contain multiple clouds (currently cumulus congestus or cumulus mediocris) at runtime.

- virtual double SILVERLINING_API GetDensityMultiplier () const

  Returns the value last set by CloudLayer::SetDensityMultiplier(), or the default value of 1.0.

- virtual void SILVERLINING_API SetCurveTowardGround (bool curve)

  Sets whether the cloud layer should curve down to meet the ground at its extents.

- virtual bool SILVERLINING_API GetCurveTowardGround () const

  Retrieves whether this cloud layer was created with SetCurveTowardGround(true).

- void SILVERLINING_API OverrideCloudColor (const Vector3 &forcedCloudColor, bool doLighting=true)

  Force the clouds in this layer to use a specific color for their directional and ambient lighting.

- void SILVERLINING_API ClearCloudColorOverride ()

  Clears usage of a specific cloud lighting color set previously with CloudLayer::OverrideCloudColor().

- bool SILVERLINING_API GetOverriddenCloudColor (Vector3 &color, bool &doLighting) const

  Returns true and populates the color parameter with the cloud color override set via CloudLayer::OverrideCloudColor(), or returns false if a cloud color override is not currently in use for this layer.

11.4.1 Detailed Description

A collection of clouds of uniform classification organized into a "deck" of similar altitudes. CloudLayer is a bse class that cannot be directly instantiated; instead, it is returned by the class factory CloudLayerFactory. Call CloudLayerFactory::Create() to obtain a CloudLayer object for a specific cloud type.

Once created, you'll want to set the CloudLayer's dimensions and density (coverage) using the SetBaseWidth(), SetBaseLength(), SetBaseAltitude(), SetThickness(), and SetDensity() methods. You will also need to position it using SetLayerPosition().

Initially, a CloudLayer will not contain any clouds. Once specifying your preferences for the layer's dimensions and density, you must then call SeedClouds() to simulate cloud growth within the cloud deck.

Once your CloudLayer object is created, configured, and seeded, you may then pass it into AtmosphericConditions::AddCloudLayer() to add it to your scene.

Once "seeded", the above methods to modify the layer's dimensions and density will have no effect - they are burned in at that point. You may, however, call SetLayerPosition() at runtime. To change the other properties of the cloud layer at runtime, you need to remove the cloud layer and re-create it.

See SetBaseWidth() for information on important differences between different types of CloudLayers.

11.4.2 Constructor & Destructor Documentation

11.4.2.1 SilverLining::CloudLayer::CloudLayer ( )

Default constructor.

This is a virtual base class, so this constructor is not directly invoked by your application. CloudLayer objects are instantiated by calling CloudLayerFactory::Create().

11.4.2.2 virtual SilverLining::CloudLayer::~CloudLayer ( ) [virtual]

Destructor.

Removes and deletes all clouds added to this CloudLayer.

11.4.3 Member Function Documentation

11.4.3.1 virtual bool SILVERLINING_API SilverLining::CloudLayer::AddCloudAt (const Atmosphere & *atm,* const Vector3 & *relativePosition,* const Vector3 & *dimensions*) [inline, virtual]

Adds an individual cloud to an already-seeded cloud layer.

Note this method will only work on cloud layers that contain many clouds, such as cumulus congestus and cumulus mediocris. If you attempt to use this on a layer that inherently contains a single cloud (such as stratus, cumulonimbus, or cirrus,) this method

will do nothing and return false. To be sure, use the CloudLayer::SupportsAddCloudAt() method to see if this method is avaialble on this cloud layer type.

All positions are relative to the cloud layer position, and must fall within the bounds defined by the cloud layer's width, depth, and height. If it does not, false will be returned and the cloud will not be added.

This method may be used to provide finer control over cloud formations and frontal boundaries. Even if you plan to manually position every cloud in the layer, you must still call CloudLayer::SeedClouds() first with a cloud coverage of 0 (via CloudLayer::SetDensity()). SeedClouds() performs some required initialization for the cloud layer. If you attempt to call AddCloudAt() on a layer that has not been seeded, again false will be returned.

Again: first call CloudLayer::SeedClouds(), then CloudLayer::AddCloudAt() repeatedly for each cloud you wish to add.

You may also mix and match procedurally-created clouds with your own clouds by setting the cloud layer coverage to a non-zero value, calling SeedClouds(), and then calling AddClouds() to add your clouds to the proceduraly generated ones.

Parameters

atm	A const reference to the Atmosphere object that will ultimately contain this CloudLayer. This atmosphere must have been initialized with Atmosphere::Initialize(), or this method will fail.
relativePosition	The center position of the cloud relative to the center of the cloud layer. Regardless of your coordinate system, this position should assume that positive Y is "up", X is along the "east" axis, and Z along the "south". These coordinates must fall within the bounds defined by the cloud layer's width and length, or this method will return false. Note that in infinite cloud layers (CloudLayer::SetIsInfinite()), each cloud may get repositioned to wrap around the edges of the layer as the camera moves. You should take care not to position clouds too closely on top of each other, or popping artifacts may result if they re-sort in a different order as the view angle changes.
dimensions	The size of this cloud. The X component represents the cloud width along the "east" dimension, Y represents the cloud height, and Z the cloud length along the "south" dimension.

Returns

True if the cloud was successfully added. False may be returned if the cloud layer has not yet been seeded with CloudLayer::SeedClouds(), if an attempt was made to place a cloud outside of the cloud layer's bounds, if an uninitialized Atmosphere was passed in, this size of the cloud is smaller than the size of a single cloud puff, or if this cloud layer does not support AddCloudAt().

11.4.3.2 void SILVERLINING_API SilverLining::CloudLayer::AddLightningListener (LightningListener * *listener*)

Adds a LightningListener - derived class to the list of LightningListeners to receive notifications of lightning strikes created by this CloudLayer.

The CloudLayer will not destroy the LightningListener; its memory management is up to the caller.

See also

ClearLightningListeners()

11.4.3.3 void SILVERLINING_API SilverLining::CloudLayer::ClearCloudColorOverride ( )

Clears usage of a specific cloud lighting color set previously with CloudLayer::OverrideCloudColor().

11.4.3.4 void SilverLining::CloudLayer::ClearClouds ( )

Removes all clouds from this cloud layer.

11.4.3.5 void SILVERLINING_API SilverLining::CloudLayer::ClearLightningListeners ( )

Clears the list of LightningListener objects that receive notifications of lightning strikes from this CloudLayer.

It does not delete the LightingListener objects.

See also

AddLightningListener()

11.4.3.6 virtual bool SILVERLINING_API SilverLining::CloudLayer::ExportToVRML (const char * *dirPath*) [virtual]

Exports each individual cloud into a numbered .WRL file within the specified directory path.

Each .WRL file will be a simple collection of translated spheres that represent the metaballs that make up each cloud.

These .WRL files may then be tweaked using a 3D graphics package, and then re-imported by placing them in resources/clouds/(cloudtype)/. (cloudtype) is the config file prefix for the settings for the given cloud layer type in resources/Atmosphere.config, for example cumulus-congestus or cumulonimbus.

If a given cloud type has .WRL files in its matching resources/clouds/ subdirectory, SilverLining will randomly select these files as the layer is initialized in SeedClouds(), instead of generating the clouds algorithmically from a cellular automata.

This will currently only produce output for cumulus cloud decks.

See also

SeedClouds()

11.4.3.7 virtual bool SILVERLINING_API SilverLining::CloudLayer::ForceLightning (bool *value* = true) [inline, virtual]

Forces a lightning strike starting with this frame on cloud layers that support lightning.

If the lightning discharge mode is FORCE_ON_OFF, the value parameter controls whether lightning is on or off.

11.4.3.8 double SILVERLINING_API SilverLining::CloudLayer::GetAlpha ( ) const [inline]

Retrieves explicit translucency set by SetAlpha().

11.4.3.9 double SILVERLINING_API SilverLining::CloudLayer::GetBaseAltitude ( ) const [inline]

Retrieves the altitude above sea level of the CloudLayer botton in meters.

11.4.3.10 double SILVERLINING_API SilverLining::CloudLayer::GetBaseLength ( ) const [inline]

Retrieves the dimension of the CloudLayer along the Z axis.

See also

SetBaseLength()

11.4.3.11 double SILVERLINING_API SilverLining::CloudLayer::GetBaseWidth ( ) const [inline]

Retrieves the dimension of the CloudLayer along the X axis.

See also

SetBaseWidth()

11.4.3.12 bool SILVERLINING_API SilverLining::CloudLayer::GetCloudWrapping ( ) const [inline]

Returns whether non-infinite cloud layer types will wrap wind-blown clouds to remain within the original bounds of the layer, or if wind will move the entire layer indefinitely.

11.4.3.13 virtual bool SILVERLINING_API SilverLining::CloudLayer::GetCurveTowardGround ( ) const [inline, virtual]

Retrieves whether this cloud layer was created with SetCurveTowardGround(true).

11.4.3.14 double SILVERLINING_API SilverLining::CloudLayer::GetDensity ( ) const [inline]

Retrieves the cloud coverage set for this CloudLayer.

Returns

The cloud coverage of this deck from 0 - 1.0.

See also

SetDensity()

11.4.3.15 virtual double SILVERLINING_API SilverLining::CloudLayer::GetDensityMultiplier ( ) const [inline, virtual]

Returns the value last set by CloudLayer::SetDensityMultiplier(), or the default value of 1.0.

11.4.3.16 bool SILVERLINING_API SilverLining::CloudLayer::GetEnabled ( ) const [inline]

Accesses if the CloudLayer is currently enabled for rendering.

See also

SetEnabled()

11.4.3.17 bool SILVERLINING_API SilverLining::CloudLayer::GetFadeTowardEdges ( ) const [inline]

Retrieves whether an infinite or wrapping cloud layer has fade effects enabled as the clouds approach the boundaries of the layer.

See also

SetFadeTowardEdges() .

11.4.3.18 bool SILVERLINING_API SilverLining::CloudLayer::GetIsInfinite ( ) const [inline]

Returns whether this cloud layer is "infinite" and attempts to remain fixed relative to the camera position as best it can.

See also

SetIsInfinite()

11.4.3.19 void SILVERLINING_API SilverLining::CloudLayer::GetLayerPosition (double & *east*, double & *south*) const [inline]

Retrieves the east and south coordinates of the CloudLayer's center, in meters.

See also

SetLayerPosition()

11.4.3.20 virtual LightningDischargeMode SILVERLINING_API SilverLining::CloudLayer::GetLightningDischargeMode ( ) [inline, virtual]

Returns the current lightning discharge mode for this cloud layer, or MAX_DISCHARGE_-MODE on layer that don't support lightning.

11.4.3.21 virtual double SILVERLINING_API SilverLining::CloudLayer::GetMaxHeight ( ) const [inline, virtual]

Retrieves the height of the bounding box enclosing this cloud layer.

For cumulus cloud types, this is different than GetThickness(), which specifies the variation in cloud base heights.

11.4.3.22 bool SILVERLINING_API SilverLining::CloudLayer::GetOverriddenCloudColor (Vector3 & *color,* bool & *doLighting*) const

Returns true and populates the color parameter with the cloud color override set via CloudLayer::OverrideCloudColor(), or returns false if a cloud color override is not currently in use for this layer.

11.4.3.23 double SILVERLINING_API SilverLining::CloudLayer::GetThickness ( ) const [inline]

Retrieves the thickness of the CloudLayer in meters.

See SetThickness() for descriptions of what this value represents for various cloud types.

11.4.3.24 CloudTypes SILVERLINING_API SilverLining::CloudLayer::GetType ( ) const [inline]

Returns the cloud type of this layer.

11.4.3.25 void SILVERLINING_API SilverLining::CloudLayer::GetWind (double & *windX,* double & *windZ*) const [inline]

Retrieves the cloud layer specific wind vector.

See also

SetWind()

11.4.3.26 virtual bool SILVERLINING_API SilverLining::CloudLayer::HasPrecipitationAtPosition (double *x,* double *y,* double *z*) const [virtual]

Returns whether a precipitation type other than NONE will be simulated at the given camera position.

If you're under a cloud and precipitation has been assigned to this cloud layer using SetPrecipitation(), this will return true. The specific effect may be retrieved with GetPrecipitation().

Parameters

x	The position, in world coordinates, for which you wish to test for precipitation effects.

y	The position, in world coordinates, for which you wish to test for precipitation effects.
z	The position, in world coordinates, for which you wish to test for precipitation effects.

11.4.3.27 bool SILVERLINING_API SilverLining::CloudLayer::Intersect (const Vector3 & *Origin*, const Vector3 & *Direction*, double & *range*) const

Test if ray intersects this cloud layer.

Currently only supported on cumulus cloud layers.

11.4.3.28 virtual bool SILVERLINING_API SilverLining::CloudLayer::IsInsideCloud (double *x*, double *y*, double *z*) const [virtual]

Returns whether the given location is inside the bounding box of any of the clouds within this cloud layer.

Parameters

x	The position, in world coordinates, for which you wish to test for a cloud intersection.
y	The position, in world coordinates, for which you wish to test for a cloud intersection.
z	The position, in world coordinates, for which you wish to test for a cloud intersection.

11.4.3.29 bool SILVERLINING_API SilverLining::CloudLayer::IsRenderable ( ) const [inline]

Returns whether this cloud layer needs to be rendered at all; if it is disabled via SetEnabled() and any fading time has expired, this will return false.

See also

SetEnabled()

11.4.3.30 void SILVERLINING_API SilverLining::CloudLayer::OverrideCloudColor (const Vector3 & *forcedCloudColor*, bool *doLighting* = true)

Force the clouds in this layer to use a specific color for their directional and ambient lighting.

Call CloudLayer::ClearCloudColorOverride() to go back to using the simulated color for the given time and location. The color passed in will be used as-is, so be sure to scale it for day vs. night for example.

Parameters

forced-CloudColor	The color you wish to force this cloud layer to become.
doLighting	Whether the color should be darkened automatically according to the time of day.

11.4.3.31 virtual bool SILVERLINING_API SilverLining::CloudLayer::Restore (const Atmosphere & *atm*, const char * *filePath*) [virtual]

Restores a cloud deck from a file previously generated with the Save() method.

It is not necessary to explicitly configure the cloud's dimensions, position, and density, or to call SeedClouds(), when Restore() is called on a newly instantiated cloud layer. Restore() offers a way to quickly load a known good cloud layer without recreating it from scratch every time using SeedClouds(). It will create the layer more quickly than SeedClouds(), but you'll lose the randomness of the clouds within the layer by using this method.

Parameters

atm	A const reference to the Atmosphere object that will ultimately contain this CloudLayer.
filePath	The fully qualified path to the file previously created with Save().

See also

Save()
SeedClouds()

Returns

True if the file was successfully restored; false if the file could not be found or was an invalid file format.

11.4.3.32 virtual bool SILVERLINING_API SilverLining::CloudLayer::Save (const char * *filePath*) [virtual]

Saves this cloud deck's clouds to a file for fast restoring later.

This is faster than calling SeedClouds to generate the deck at runtime, and also allows a way to reproduce the same cloud decks by bypassing the randomness of SeedClouds.

Parameters

filePath	The fully qualified path to the file you wish to save this cloud deck's configuration to.

See also

SeedClouds()
Restore()

Returns

True if the file was successfully written.

11.4.3.33 virtual bool SILVERLINING_API SilverLining::CloudLayer::SeedClouds (const Atmosphere & *atm*) [pure virtual]

Generates clouds within the cloud layer.

SeedClouds() may only be called after configuring the CloudLayer by calling the SetBaseWidth(), SetBaseLength(), SetBaseAltitude(), SetThickness(), SetDensity(), and SetLayerPosition() methods. When called, it will invoke a cloud growth model to fill the cloud layer with clouds to meet the specified properties of the CloudLayer.

Once a cloud layer is seeded, you may use the Save() method to save it to disk if you'd like to be able to quickly restore it with the Restore() method, instead of recreating the layer from scratch every time.

No clouds will appear within your CloudLayer unless SeedClouds() is called!

Parameters

atm	A const reference to the Atmosphere object that will ultimately contain this CloudLayer. This atmosphere must have been initialized with Atmosphere::Initialize(), or this method will fail.

See also

Save()
Restore();

11.4.3.34 virtual bool SILVERLINING_API SilverLining::CloudLayer::Serialize (std::ostream & *stream*) [virtual]

Flattens this object and everything in it to a stream buffer.

11.4.3.35 void SILVERLINING_API SilverLining::CloudLayer::SetAlpha (double *pAlpha*) [inline]

Sets translucency for the cloud layer; 0 = transparent, 1.0 = normal opacity.

Default is 1.0. This can be used to give clouds a softer appearance and blend into the background better.

11.4.3.36 virtual void SILVERLINING_API SilverLining::CloudLayer::SetBaseAltitude (double *meters*, bool *updateCloudPositions* = true) [virtual]

Sets the altitude above sea level of the bottom of the CloudLayer.

Here's some guidance on realistic settings (in meters) for various CloudLayer types:

- Cumulus congestus, cumulus mediocris: Typically found fairly low; between 1000-4000m.
- Cumulonimbus: Generally extends to the ground if it's raining. 400m is an appropriate setting.
- Stratus: Found very low; typically 500-2000 m.
- Cirrus: Found very high; 6000 m or higher.
- Sandstorm: Set to the local ground altitude to position the sandstorm on the ground.

SetBaseAltitude() may be called at runtime after a cloud layer has been seeded to change its position, and should be called before seeding the cloud layer.

The updateCloudPositions indicates whether the cloud altitudes within this layer should be updated immediately. If set to false, individual clouds will not update until they are repositioned by other means, such as wrapping around an infinite cloud layer or a call to SeedClouds().

11.4.3.37 virtual void SILVERLINING_API SilverLining::CloudLayer::SetBaseLength (double *meters*) [inline, virtual]

Sets the size of the CloudLayer along the Z axis.

See SetBaseWidth() for guidance on appropriate values for different types of CloudLayers.

11.4.3.38 virtual void SILVERLINING_API SilverLining::CloudLayer::SetBaseWidth (double *meters*) [inline, virtual]

Sets the size of the CloudLayer along the X axis.

The cloud layer must be at least as big as the size of individual clouds within it. The size of the cloud layer may have important performance and visual impacts, so it should be chosen with care. Here are some general guidelines for setting the cloud layer dimensions:

- Cumulus congestus, mediocris, and sandstorms: The number of clouds created is proportional to the square of the layer dimensions, so performance will degrade quickly if unreasonably high dimensions are chosen. A balance must be found between performance (not having too many clouds to render at real-time speed) and visual acceptability (having enough clouds to fill the scene.) Dimensions of around 30 km or so generally fulfill both demands, but the number of clouds also depends on the cloud layer density (see SetDensity()).

- Stratus: The width represents the size of the entire stratus cloud layer. If this is an infinite stratus layer, it represents how far around the camera the layer will extend to, and you'll want to keep this at some large number like 100 km.

- Cumulonimbus: Due to the large size of cumulonimbus clouds and their isolated organization along "squall lines", it's not realistic to have a rectangular deck of cumulonimbus clouds. Instead, the size of the cumulonimbus cloud layer will reflect the size of a single cumulonimbus cloud that fills the entire layer. Choose dimensions that reflect the size of the cloud itself, typically on the order of 5 km.

- Cirrus: Cirrus decks are rendered as a high, textured plane, but it does not extend infinitely like stratus decks do. A large enough dimension should be chosen to fill the sky of your scene at the deck's altitude; typically, 100,000 meters is appropriate.

- Stratocumulus: Stratocumulus clouds are rendered using 3D volumetric data that can be very memory-intensive. Cloud layer sizes above 50,000 meters are not recommended, and will also result in sampling artifacts. This upper bound is enforced by the stratocumulus-max-size setting in resources/silverlining.config.

11.4.3.39 virtual void SILVERLINING_API SilverLining::CloudLayer::SetCloudAnimationEffects (double *voxelSpinRate*, bool *enableGrowth*, int *initialGrowthIterations* = 0, int *timeStepInterval* = 0) [inline, virtual]

Controls cloud animation effects over time.

This only affects cumulus cloud types. Must be called prior to CloudLayer::SeedClouds().

Parameters

voxelSpinRate	For cumulus clouds, the maximum rate at which individual cloud voxels rotate to simulate convection in the cloud. Specified in radians per second. Set to 0 for no animation.
enableGrowth	Controls whether the shape of the cloud itself evolves over time, via cellular automata techniques. This can cause clouds to "grow" over time, but comes with a performance cost.
initialGrowthIterations	If enableGrowth is true, this controls the initial number of iterations of the cullular automata to run to form each cloud prior to the first frame. If you leave this set to zero, the default vaule for the specific cloud type will be used, resulting in a fully-formed cloud at the outset. However, you could set this as low as 1, resulting in clouds growing over time from very small clouds.
timeStepInterval	The time between cellular automata iterations, in seconds. Changes will be interpolated across this time. Longer times will result in slower cloud formation. To use the default settings for this cloud type, use a value of 0.

11.4.3.40 void SILVERLINING_API SilverLining::CloudLayer::SetCloudWrapping (bool *wrap*) [inline]

Infinite cloud layers will always wrap the clouds around the camera position, but you can also enable this behavior relative to the position of a non-infinite cloud layer.

This means that as the wind blows individual clouds, they will remain within the original bounds of the cloud layer rather than allowing the entire layer to move with the wind. Cirrus, cirrocumulus, and cumulonimbus cloud layers cannot be wrapped. This mode is false by default.

See also

GetCloudWrapping()

11.4.3.41 virtual void SILVERLINING_API SilverLining::CloudLayer::SetCurveTowardGround (bool *curve*) [inline, virtual]

Sets whether the cloud layer should curve down to meet the ground at its extents.

Can be useful for creating the illusion of cloud layers that are bigger than they really are. Generally this should be called before seeding the cloud layer. Default is false. With cumulus clouds, this effect is strongest when the camera is near the ground (as determined from the altitude passed in via the Location object) and goes away when the altitude reaches the cloud layer's base altitude.

11.4.3.42 virtual void SILVERLINING_API SilverLining::CloudLayer::SetDensity (double *coveragePercentage*) [inline, virtual]

Sets the percentage of the sky that should be covered by this CloudLayer, from 0 - 1.0.

For cumulus congestus and cumulus mediocris decks, this will affect the number of clouds generated during SeedClouds(), within a coverage tolerance of 5%. For stratus decks, setting a value less than 1.0 will result in a "broken" cloud deck, which results in extremely high performance characteristics for variable cloud clover, but is much less visually realistic than a cumulus congestus deck. This value is ignored for cirrus and cumulonimbus cloud decks, which always represent a single cloud.

Parameters

coveragePercentage	The amount of the sky that should be covered by clouds, from 0 - 1.0.

11.4.3.43 virtual void SILVERLINING_API SilverLining::CloudLayer::SetDensityMultiplier (double *multiplier*) [inline, virtual]

This allows you to reduce the coverage of cloud layers that contain multiple clouds (currently cumulus congestus or cumulus mediocris) at runtime.

The cloud layer will initially create enough clouds in the scene to achieve the coverage specified in CloudLayer::SetDensity(). By calling CloudLayer::SetDensityMultiplier, we'll randomly fade some clouds in or out to reduce the coverage from the original density. The clouds you don't see will still be using memory, so use this technique with care and only if you need to vary the layer density continuously.

So, one technique would be to create a cloud layer with SetDensity(1.0), and then use SetDensityMultiplier() to change the density to whatever you want at runtime.

Parameters

multiplier	A value from 0 to 1.0 controlling what percentage of this cloud layer's clouds will actually be displayed.

11.4.3.44 void SILVERLINING_API SilverLining::CloudLayer::SetEnabled (bool *enabled*, unsigned long *fadeTimeMS* = 0)

Enable rendering of this cloud layer.

New cloud layers are enabled by default.

Why would you want to explicitly enable and disable a cloud layer? Well, creating and

seeding cloud layers - especially cumulus cloud layers - is computationally expensive and can take up to a second or two. If your application requires several cloud layers that are displayed at different times, it may make sense to create them all when your application initializes, and then just enable and disable them as needed, which is a much faster operation.

Optionally, you may fade cloud layers in or out over time to smoothly introduce them into your scene. Specify fadeTimeMS in order to enable this effect when enabling or disabling a cloud layer. Fading effects on cumulus layers are only supported on systems that support vertex shaders.

Parameters

enabled	True if this cloud layer should be rendered during Atmosphere::DrawObjects(), false if it should not be rendered.
fadeTimeMS	The amount of time, in milliseconds, for the cloud layer to fade in or out.

11.4.3.45 void SILVERLINING_API SilverLining::CloudLayer::SetFadeTowardEdges (bool *fade*) [inline]

For infinite cloud layers or cloud layers with "wrapping" enabled this option allows you to automatically fade clouds out as they approach the boundary of the cloud layer where they will pop out and reappear on the opposite side of the layer.

This fading hides the popping in the distance.

See also

SetCloudWrapping()
SetIsInfinite()

11.4.3.46 virtual void SILVERLINING_API SilverLining::CloudLayer::SetIsInfinite (bool *inf*) [inline, virtual]

Selects whether this cloud layer is "infinite" - that is, it will always surround the camera when feasible, and never gets blown away by wind.

For layers of cumulus clouds, individual clouds will wrap around the boundaries of the layer to keep the layer centered at the camera position at all times.

The specific effect of "infinite" varies by the cloud layer type. Cumulus congestus and mediocris cloud layers behave as described above, where individual clouds wrap around the cloud layer's boundaries relative to the camera position. For cirrus and cirrocumulus clouds, setting them to "infinite" means they are not affected by wind and remain fixed above the camera position. Infinite stratus cloud layers remain positioned

relative to the camera position and simulate relative motion by rotating its texture coordinates. This method does not affect cumulonimbus cloud layers, as they only contain a single cloud.

Be default, cloud layers are not infinite and must be positioned using SetLayerPosition().

See also

GetIsInfinite()

11.4.3.47 void SILVERLINING_API SilverLining::CloudLayer::SetLayerPosition (double *eastCoord,* double *southCoord*)

Specifies the location of the center of the CloudLayer.

Sets the "East" and "South" coordinates of the CloudLayer's center position. For a right-handed coordinate system where Z points in, you'd pass in the X and Z coordinates of where you want the cloudlayer to be centered over, for example. A right-handed coordinate system where Z points up would pass in X and -Y instead.

Cloud positions may move over time in response to simulated wind (see AtmosphericConditions::SetWind()).

If you are using a custom set of basis vectors, be sure that you have called Atmosphere::SetUpVector() and Atmosphere::SetRightVector() before calling this method. The cloud layer will be oriented according to the basis defined by the current up and right vectors when SetLayerPosition() is called. If you're in a geocentric, round-Earth system, you'll want to set the up and right vectors to reflect the local basis at the cloud's location, and then set these back to reflect the eyepoint before rendering with Atmosphere::DrawSky() and Atmosphere::DrawObjects().

Calling SetLayerPosition will reset any accumulated wind drift as well.

SetLayerPosition() and SetBaseAltitude() are the only methods on a CloudLayer that you may change at runtime, after the CloudLayer has been seeded and added into the scene. Modifying other properties of the CloudLayer after it's been seeded require removing the CloudLayer from your AtmosphericConditions, and re-creating the layer.

This method is not needed with infinite cloud layers, which are always positioned relative to the camera.

Parameters

eastCoord	The east coordinate of the CloudLayer's center position, in meters.
southCoord	The south coordinate of the CloudLayer's center position, in meters.

11.4.3.48 virtual void SILVERLINING_API SilverLining::CloudLayer::SetLightningDischargeMode (LightningDischargeMode *mode*) [inline, virtual]

Sets the lightning mode.

AUTO_DISCHARGE will fire lightning automatically at random. FORCE_ON_OFF gives the application explicit control via CloudLayer::ForceLightning(). FIRE_AND_FORGET allows lightning to be initiated via CloudLayer::ForceLightning(), but the lightning strike will then finish automatically.

11.4.3.49 void SILVERLINING_API SilverLining::CloudLayer::SetPrecipitation (int *type*, double *intensity*, double *nearClip* = -1.0, double *farClip* = -1.0, bool *useDepthBuffer* = false)

Simulates precipitation under this cloud layer of a specified type.

Any time the camera is under a cloud of this layer, precipitation effects will display if the type is set to something other than NONE.

Note, you may also set global precipitation effects that are not associated with a CloudLayer with the similar AtmosphericConditions::SetPrecipitation() method.

For mixed precipitation, you may call SetPrecipitation multiple times with different precipitation types. To clear all precipitation, call SetPrecipitation with a type of NONE. If you call this method multiple times for the same precipitation type, the intensity specified will overwrite the intensity previously specified for that type.

Parameters

type	The type of precipitation to simulate under this cloud layer - NONE, RAIN, WET_SNOW, DRY_SNOW, or SLEET.
intensity	The simulated rate of precipitation, in millimeters per hour. Reasonable ranges might be between 1 for light rain or 20 for heavier rain. This value will be clamped to the value specified by rain-max-intensity, snow-max-intensity, or sleet-max-intensity in resources/SilverLining.config, which is 30 by default.
nearClip	How close to the camera the closest precipitation particles will be rendered. The near clipping plane will be adjusted to include this distance while the precipitation is being rendererd. Values less than zero will result in the default value being applied.
farClip	The farthest distance from the camera that precipitation particles will be rendered. Since there is an upper bound on the number of particles rendered per frame, changing the difference between nearClip and farClip may result in changes to the intensity of the precipitation. By default this difference is 7.5 world units. Values less than zero will result in the default value being applied.
useDepthBuffer	Set this to true if you want to enable depth buffer testing of precipitation particles against your scene.

See also

HasPrecipitationAtPosition()
GetPrecipitation()

11.4.3.50 virtual void SILVERLINING_API SilverLining::CloudLayer::SetThickness (double *meters*) [inline, virtual]

Sets the thickness of the CloudLayer in meters.

"Thickness" has different meanings for different CloudLayer types:

- Cumulus congestus, mediocris: The thickness specifies the variance in the altitude of the bottoms of each individual cloud within the CloudLayer. The actual height of each cloud is determined procedurally as part of the cloud growth model, and is not affected by SetThickness(). So, setting the thickness to 0 would result in a layer of clouds that all start at the same base altitude; setting a thickness of 100 would provide for some variance for increased realism. In reality, cloud layers rest on top of an air mass that cause most of the clouds to have flattened bottoms that start at roughly the same altitude.

- Cumulonimbus: SetThickness() is unused for cumulonimbus clouds. The height of the cloud will be determined proportionally from the setting in SetBaseWidth().

- Stratus: The thickness in this case specifies the actual thickness of the stratus deck, that is, the distance between the top and bottom of the stratus layer. Stratus clouds vary in thickness; 1000-2000 m are reasonable values. The thickness of the stratus deck will not impact performance, unlike cumulus decks.

- Stratocumulus: Like stratus, the thickness in this case specifies the actual thickness of the stratocumulus layer. This needs to be thick enough to contain the clouds within this layer; typically 3000 meters should suffice. If it's too small, the clouds will be unrealistically small and also transmit too much sunlight, so set this with care.

- Cirrus: The thickness is ignored for cirrus decks; cirrus decks are modeled as infinitely thin layers of ice crystals at high altitude.

- Sandstorm: Pass 0 to ensure a flat bottom to the sandstorm, or just leave this unspecified.

11.4.3.51 void SILVERLINING_API SilverLining::CloudLayer::SetType (CloudTypes *type*) [inline]

Sets the type of this cloud layer.

11.4.3.52 void SILVERLINING_API SilverLining::CloudLayer::SetWind (double *windX*, double *windZ*) [inline]

Sets a cloud layer specific wind vector which is additive to any global wind set at the AtmosphericConditions level.

The vector itself specifies the direction the wind is coming from, and the length of the vector provides the wind velocity.

Parameters

windX	The east vector of the CloudLayer's specific wind, in meters/s.
windZ	The south vector of the CloudLayer's specific wind, in meters/s.

See also

GetWind()

11.4.3.53 const SilverLining::CloudLayer::SL_MAP (int , double) const [inline]

Retrieves a map of precipitation types to intensities in millimeters per hour that are associated with this cloud layer.

Precipitation types include RAIN, WET_SNOW, DRY_SNOW, or SLEET.

See also

SetPrecipitation()
HasPrecipitationAtPosition()

11.4.3.54 virtual bool SILVERLINING_API SilverLining::CloudLayer::SupportsAddCloudAt ( ) const [inline, virtual]

Indicates whether this cloud layer supports the CloudLayer::AddCloudAt() method for adding user-positioned clouds to the procedural clouds.

The documentation for this class was generated from the following file:

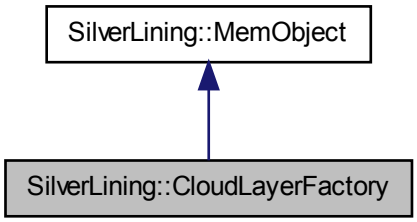
- C:/sl/Public Headers/CloudLayer.h

11.5 SilverLining::CloudLayerFactory Class Reference

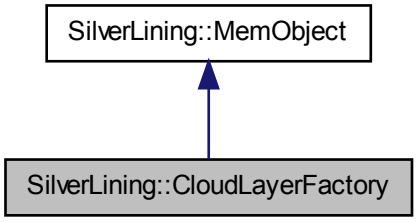
A class factory for CloudLayer objects.

`#include <CloudLayerFactory.h>`

Inheritance diagram for SilverLining::CloudLayerFactory:



Collaboration diagram for SilverLining::CloudLayerFactory:



Static Public Member Functions

- static CloudLayer *SILVERLINING_API Create (CloudTypes layerType)

  Instantiate a CloudLayer for a given cloud type.

- static bool SILVERLINING_API Serialize (CloudLayer *layer, std::ostream &stream)

  Flattens a cloud layer to a stream buffer.

- static CloudLayer *SILVERLINING_API Unserialize (const Atmosphere &atm, std::istream &stream)

  Restores a cloud layer from a stream buffer written with Serialize()

11.5.1 Detailed Description

A class factory for CloudLayer objects. Call CloudLayerFactory::Create to instantiate a CloudLayer object, which may then be configured, seeded, and submitted to the AtmosphericConditions. CloudLayer is a virtual base class, so this is the only mechanism for creating specific cloud layer types.

11.5.2 Member Function Documentation

11.5.2.1 static CloudLayer* SILVERLINING_API SilverLining::CloudLayerFactory::Create (CloudTypes *layerType*) [static]

Instantiate a CloudLayer for a given cloud type.

Currently supported CloudTypes include:

- CloudTypes::CIRROCUMULUS - High planar cloud puffs.
- CloudTypes::CIRRUS_FIBRATUS - High, wispy cirrus clouds.
- CloudTypes::STRATUS - Low clouds represented as a slab.
- CloudTypes::CUMULONIMBUS_CAPPILATUS - A large thunderhead with lightning and rain.
- CloudTypes::CUMULUS_MEDIOCRIS - Small, puffy clouds. Use sparingly.
- CloudTypes::CUMULUS_CONGESTUS - Larger cumulus clouds with flattened bottoms, optimized for performance.
- CloudTypes::CUMULUS_CONGESTUS_HI_RES - Larger cumulus clouds with flattened bottoms, optimized for appearance.
- CloudTypes::TOWERING_CUMULUS - Large cumulus clouds with strong vertical growth - basically cumulus congestus clouds growing into cumulonimbus.
- CloudTypes::STRATOCUMULUS - Low, dense, puffy clouds with some sun breaks between them.
- CloudTypes::SANDSTORM - A "haboob" cloud of dust intended to be positioned at ground level.

Parameters

layerType	The type of cloud deck to create, must be of the CloudTypes enumeration.

Returns

A pointer to a new CloudLayer for the specified type.

11.5.2.2 static bool SILVERLINING_API SilverLining::CloudLayerFactory::Serialize (CloudLayer * *layer*, std::ostream & *stream*) [static]

Flattens a cloud layer to a stream buffer.

The documentation for this class was generated from the following file:

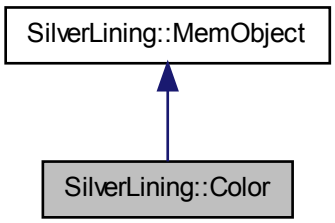
- C:/sl/Public Headers/CloudLayerFactory.h

11.6 SilverLining::Color Class Reference

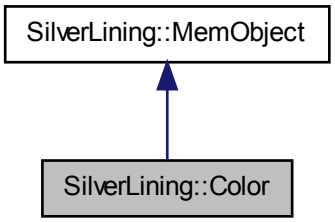
A RGBA color, where each component is represented as a float from 0-1.

`#include <Color.h>`

Inheritance diagram for SilverLining::Color:



Collaboration diagram for SilverLining::Color:



Public Member Functions

- Color ()

  Default constructor, initializes to black with no translucency.

- Color (float red, float green, float blue, float alpha)

  Constructor that takes in RGBA single-precision values that range from 0-1.

- Color (float red, float green, float blue)

  Constructor that takes in RGB single-precision values that range from 0-1.

- Color (double red, double green, double blue, double alpha)

  Constructor that takes in RGBA double-precision values that range from 0-1.

- Color (double red, double green, double blue)

  Constructor that takes in RGB double-precision values that range from 0-1.

- Color (int red, int green, int blue, int alpha)

  Constructor that takes in RGBA integer values that range from 0-1.

- Color (int red, int green, int blue)

  Constructor that takes in RGB integer values that range from 0-1.

- Color SILVERLINING_API operator∗ (float f) const

  Multiplication operator; multiplies each rgb component of the Color by a constant.

- Color SILVERLINING_API operator∗ (const Color &c) const

  Multiplication operator; multiplies two Colors together on a per-component basis.

- Color SILVERLINING_API operator+ (const Color &c) const

  Addition operator; adds two colors together on a per-component basis.

- bool SILVERLINING_API operator== (const Color &c) const

  Equality operator.

- bool SILVERLINING_API operator!= (const Color &c) const

  Inequality operator.

- Vector4 SILVERLINING_API ToVector4 () const

  Convert Color to Vector4.

- Vector3 SILVERLINING_API ToVector3 () const

  Convert Color to Vector3.

- void SILVERLINING_API ScaleToUnitOrLess ()

  If any component of the Color exceeds 1.0, every component will be scaled down uniformly such that the maximum color component equals 1.0.

- void SILVERLINING_API ScaleToValueOrLess (float value)

  If any component of the Color exceeds the given value, every component will be scaled down uniformly such that the maximum color component equals it.

- void SILVERLINING_API ClampToUnitOrLess ()

  Clamps each color component to be within the range [0, 1.0].

- Color SILVERLINING_API ToGrayscale () const

  Convert the RGB color to a grayscale value.

- void SILVERLINING_API Serialize (std::ostream &s) const

  Save this color to disk.

- void SILVERLINING_API Unserialize (std::istream &s)

  Restore this color from disk.

11.6.1 Detailed Description

A RGBA color, where each component is represented as a float from 0-1.

11.6.2 Constructor & Destructor Documentation

11.6.2.1 SilverLining::Color::Color ( ) [inline]

Default constructor, initializes to black with no translucency.

11.6.2.2 SilverLining::Color::Color (float *red*, float *green*, float *blue*, float *alpha*) [inline]

Constructor that takes in RGBA single-precision values that range from 0-1.

11.6.2.3 SilverLining::Color::Color (float *red*, float *green*, float *blue*) [inline]

Constructor that takes in RGB single-precision values that range from 0-1.

The alpha component is assumed to be 1.0.

11.6.2.4 SilverLining::Color::Color (double *red*, double *green*, double *blue*, double *alpha*) [inline]

Constructor that takes in RGBA double-precision values that range from 0-1.

11.6.2.5 SilverLining::Color::Color (double *red*, double *green*, double *blue*) [inline]

Constructor that takes in RGB double-precision values that range from 0-1.

The alpha component is assumed to be 1.0.

11.6.2.6 SilverLining::Color::Color (int *red*, int *green*, int *blue*, int *alpha*) [inline]

Constructor that takes in RGBA integer values that range from 0-1.

11.6.2.7 SilverLining::Color::Color (int *red*, int *green*, int *blue*) [inline]

Constructor that takes in RGB integer values that range from 0-1.

The alpha component is assumed to be 1.0.

11.6.3 Member Function Documentation

11.6.3.1 void SILVERLINING_API SilverLining::Color::ClampToUnitOrLess ( ) [inline]

Clamps each color component to be within the range [0, 1.0].

11.6.3.2 Color SILVERLINING_API SilverLining::Color::operator* (float *f*) const [inline]

Multiplication operator; multiplies each rgb component of the Color by a constant.

11.6.3.3 Color SILVERLINING_API SilverLining::Color::operator* (const Color & *c*) const [inline]

Multiplication operator; multiplies two Colors together on a per-component basis.

11.6.3.4 Color SILVERLINING_API SilverLining::Color::operator+ (const Color & *c*) const [inline]

Addition operator; adds two colors together on a per-component basis.

11.6.3.5 void SILVERLINING_API SilverLining::Color::ScaleToUnitOrLess ( ) [inline]

If any component of the Color exceeds 1.0, every component will be scaled down uniformly such that the maximum color component equals 1.0.

11.6.3.6 void SILVERLINING_API SilverLining::Color::ScaleToValueOrLess (float *value*) [inline]

If any component of the Color exceeds the given value, every component will be scaled down uniformly such that the maximum color component equals it.

11.6.3.7 void SILVERLINING_API SilverLining::Color::Serialize (std::ostream & *s*) const [inline]

Save this color to disk.

11.6.3.8 Color SILVERLINING_API SilverLining::Color::ToGrayscale ( ) const [inline]

Convert the RGB color to a grayscale value.

11.6.3.9 void SILVERLINING_API SilverLining::Color::Unserialize (std::istream & *s*) [inline]

Restore this color from disk.

The documentation for this class was generated from the following file:

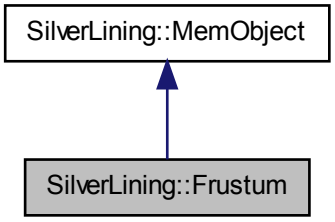
- C:/sl/Public Headers/Color.h

11.7 SilverLining::Frustum Class Reference

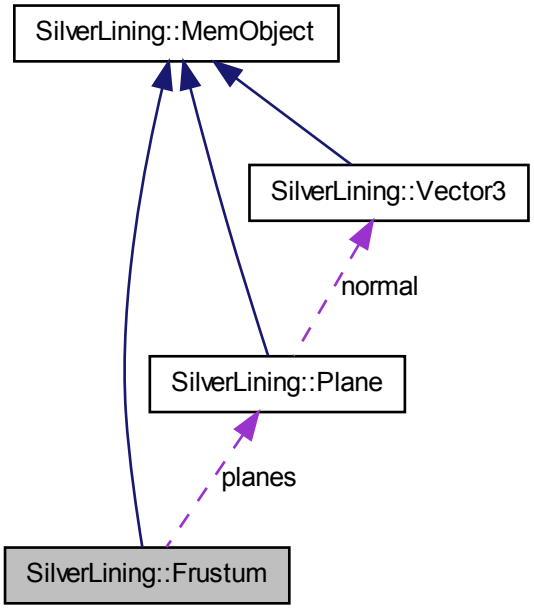
A simple class that just collects six planes together and calls it a frustum.

`#include <Frustum.h>`

Inheritance diagram for SilverLining::Frustum:



Collaboration diagram for SilverLining::Frustum:



Public Types

- enum Planes

  Identifiers for each of the six planes that make up the viewing frustum.

Public Member Functions

- Frustum ()

  Default constructor.

- ~Frustum ()

  Destructor; does nothing.

- void SILVERLINING_API SetPlane (int planeNum, const Plane &plane)

  Assigns a plane to one of the enumerated planes of the frustum.

- const Plane &SILVERLINING_API GetPlane (int planeNum) const

  Retrieves a specific Plane of the frustum.

- void SILVERLINING_API EnableFarClipCulling (bool bEnable)

  Enables culling against the far clip plane from this frustum (enabled by default.)

11.7.1 Detailed Description

A simple class that just collects six planes together and calls it a frustum.

11.7.2 Member Enumeration Documentation

11.7.2.1 enum SilverLining::Frustum::Planes

Identifiers for each of the six planes that make up the viewing frustum.

11.7.3 Constructor & Destructor Documentation

11.7.3.1 SilverLining::Frustum::Frustum ( ) [inline]

Default constructor.

11.7.3.2 SilverLining::Frustum::~Frustum ( ) [inline]

Destructor; does nothing.

11.7.4 Member Function Documentation

11.7.4.1 const Plane& SILVERLINING_API SilverLining::Frustum::GetPlane (int *planeNum*) const [inline]

Retrieves a specific Plane of the frustum.

Parameters

planeNum	A member of the Planes enumeration, specifying if you wish to retrieve the near, left, right, top, bottom, or back frustum plane.

Returns

A const reference to the Plane object requested.

11.7.4.2 void SILVERLINING_API SilverLining::Frustum::SetPlane (int *planeNum,* const Plane & *plane*) [inline]

Assigns a plane to one of the enumerated planes of the frustum.

Parameters

planeNum	A member of the Planes enumeration, specifying if this plane represents the near, left, right, top, bottom, or back frustum plane.
plane	A Plane object representing the Frustum plane specified.

The documentation for this class was generated from the following file:

- C:/sl/Public Headers/Frustum.h

11.8 SilverLining::LightningListener Class Reference

Extend this class in order to receive notifications of lightning strike events from cumulonimbus cloud layers.

`#include <LightningListener.h>`

Public Member Functions

- virtual ~LightningListener ()

  Virtual destructor.

- virtual void SILVERLINING_API LightningStartedEvent (double lightningPosX, double lightningPosY, double lightningPosZ)=0

  This pure virtual method will be called any time a lightning strike begins within a CloudLayer.

11.8.1 Detailed Description

Extend this class in order to receive notifications of lightning strike events from cumulonimbus cloud layers.

11.8.2 Constructor & Destructor Documentation

11.8.2.1 virtual SilverLining::LightningListener::~LightningListener ( ) [inline, virtual]

Virtual destructor.

11.8.3 Member Function Documentation

11.8.3.1 virtual void SILVERLINING_API SilverLining::LightningListener::LightningStartedEvent (double *lightningPosX*, double *lightningPosY*, double *lightningPosZ*) [pure virtual]

This pure virtual method will be called any time a lightning strike begins within a CloudLayer.

You must pass your LightningListener object's pointer into CloudLayer::AddLightningListener() in order to receive lightning events from a given CloudLayer. Presently, only CumulonimbusCloudLayer objects will fire these events. The position returned is the location of the lightning strike's origin, in world units.

The documentation for this class was generated from the following file:

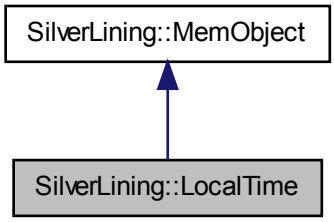
- C:/sl/Public Headers/LightningListener.h

11.9 SilverLining::LocalTime Class Reference

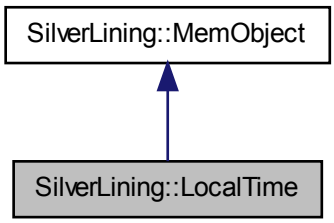
A class to represent the simulated local time, time zone, and daylight savings time observation.

`#include <LocalTime.h>`

Inheritance diagram for SilverLining::LocalTime:



Collaboration diagram for SilverLining::LocalTime:



Public Member Functions

- LocalTime ()

  Default constructor.

- virtual ~LocalTime ()

  Destructor.

- void SILVERLINING_API SetFromEpochSeconds (time_t time)

Populates the LocalTime class based on epoch seconds (ie, as returned from `time(NULL)` *).*

- void SILVERLINING_API SetYear (int year)

  Set the calendar year to simulate.

- int SILVERLINING_API GetYear () const

  Retrieves the Gregorian calendar year.

- void SILVERLINING_API SetMonth (int month)

  Sets the calendar month to simulate.

- int SILVERLINING_API GetMonth () const

  Retrieves the calendar month (1-12).

- void SILVERLINING_API SetDay (int day)

  Sets the day of the month to simulate.

- int SILVERLINING_API GetDay () const

  Retrieves the day of the month (1-31).

- void SILVERLINING_API SetHour (int hour)

  Sets the hour of day to simulate.

- int SILVERLINING_API GetHour () const

  Retrieves the hour of day (0-23).

- void SILVERLINING_API SetMinutes (int minutes)

  Sets the minute of the hour to simulate.

- int SILVERLINING_API GetMinutes () const

  Retrieves the minute of the hour (0-59).

- void SILVERLINING_API SetSeconds (double seconds)

  Sets the second of the minute to simulate.

- double SILVERLINING_API GetSeconds () const

  Retrieves the second of the minute (0-59).

- void SILVERLINING_API SetObservingDaylightSavingsTime (bool inDST)

  Sets if Daylight Savings Time is currently observed in the simulation.

- bool SILVERLINING_API GetObservingDaylightSavingsTime () const

Retrieves if daylight savings time is observed in the simulation.

- void SILVERLINING_API SetTimeZone (double zone)

  Specifies the time zone being simulated.

- double SILVERLINING_API GetTimeZone () const

  Retrieves the currently simulated time zone.

- double SILVERLINING_API GetJulianDate (bool terrestrialTime) const

  Retrieves the Julian Date that this LocalTime object represents.

- double SILVERLINING_API GetEpoch2000Centuries (bool terrestrialTime) const

  Obtains centuries and fraction since January 1, 2000 represented by this object.

- double SILVERLINING_API GetEpoch1990Days (bool terrestrialTime) const

  Obtains days elapsed since January 1, 1990 represented by this object on the Julian calendar.

- void SILVERLINING_API SetFromSystemTime ()

  Populate the object based on the system's current local time settings.

- void SILVERLINING_API AddSeconds (long seconds)

  Add the given number of seconds to the time represented by this object.

- bool SILVERLINING_API Serialize (std::ostream &stream)

  Flattens this object and everything in it to a stream buffer.

- bool SILVERLINING_API Unserialize (std::istream &stream)

  Restores this object from the stream created using Serialize()

11.9.1 Detailed Description

A class to represent the simulated local time, time zone, and daylight savings time observation. LocalTime objects are passed into AtmosphericConditions::SetTime() to change the simulated time of day. Be sure that your Location object represents a location consistent with the time zone you have specified as well, unless you choose to use GMT or another non-local time zone consistently.

11.9.2 Constructor & Destructor Documentation

11.9.2.1 SilverLining::LocalTime::LocalTime ( )

Default constructor.

Creates a local time with default settings (read from the SilverLining.config file if constructed after calling Atmosphere::Initialize()).

11.9.2.2 virtual SilverLining::LocalTime::~LocalTime ( ) [inline, virtual]

Destructor.

11.9.3 Member Function Documentation

11.9.3.1 void SILVERLINING_API SilverLining::LocalTime::AddSeconds (long *seconds*)

Add the given number of seconds to the time represented by this object.

11.9.3.2 int SILVERLINING_API SilverLining::LocalTime::GetDay ( ) const [inline]

Retrieves the day of the month (1-31).

See also

SetDay()

11.9.3.3 double SILVERLINING_API SilverLining::LocalTime::GetEpoch1990Days (bool *terrestrialTime*) const

Obtains days elapsed since January 1, 1990 represented by this object on the Julian calendar.

Used for internal astronomical calculations. Since this number is smaller than that returned by GetJulianDate(), it is of higher precision.

Parameters

terrestrialTime	Specifying terrestrial time means you want atomic clock time, not corrected by leap seconds to account for slowing of the Earth's rotation, as opposed to GMT which does account for leap seconds.

Returns

The fractional number of days elapsed since January 1, 1990.

11.9.3.4 double SILVERLINING_API SilverLining::LocalTime::GetEpoch2000Centuries (bool *terrestrialTime*) const

Obtains centuries and fraction since January 1, 2000 represented by this object.

Used for internal astronomical calculations. Since this number is smaller than that returned by GetJulianDate(), it is of higher precision.

Parameters

terrestrial-Time	Specifying terrestrial time means you want atomic clock time, not corrected by leap seconds to account for slowing of the Earth's rotation, as opposed to GMT which does account for leap seconds.

Returns

The fractional number of centuries elapsed since January 1, 2000.

11.9.3.5 int SILVERLINING_API SilverLining::LocalTime::GetHour ( ) const [inline]

Retrieves the hour of day (0-23).

See also

SetHour()

11.9.3.6 double SILVERLINING_API SilverLining::LocalTime::GetJulianDate (bool *terrestrialTime*) const

Retrieves the Julian Date that this LocalTime object represents.

Julian Dates are used for astronomical calculations (such as our own ephemeris model) and represent days and fractions since noon Universal Time on January 1, 4713 BCE on the Julian calendar. Note that due to precision limitations of 64-bit doubles, the resolution of the date returned may be as low as within 8 hours.

Parameters

terrestrial-Time	Specifying terrestrial time means you want atomic clock time, not corrected by leap seconds to account for slowing of the Earth's rotation, as opposed to GMT which does account for leap seconds.

Returns

the Julian date that this object represents.

11.9.3.7 int SILVERLINING_API SilverLining::LocalTime::GetMinutes ( ) const [inline]

Retrieves the minute of the hour (0-59).

See also

SetMinutes()

11.9.3.8 int SILVERLINING_API SilverLining::LocalTime::GetMonth ( ) const [inline]

Retrieves the calendar month (1-12).

See also

SetMonth()

11.9.3.9 bool SILVERLINING_API SilverLining::LocalTime::GetObservingDaylightSavingsTime ( ) const [inline]

Retrieves if daylight savings time is observed in the simulation.

See also

SetObservingDaylightSavingsTime()

11.9.3.10 double SILVERLINING_API SilverLining::LocalTime::GetSeconds ( ) const [inline]

Retrieves the second of the minute (0-59).

See also

SetSeconds()

11.9.3.11 double SILVERLINING_API SilverLining::LocalTime::GetTimeZone ( ) const [inline]

Retrieves the currently simulated time zone.

Returns

The hour offset from GMT, ignoring daylight savings time.

See also

SetTimeZone()

11.9.3.12 int SILVERLINING_API SilverLining::LocalTime::GetYear ( ) const [inline]

Retrieves the Gregorian calendar year.

See also

SetYear()

11.9.3.13 bool SILVERLINING_API SilverLining::LocalTime::Serialize (std::ostream & *stream*)

Flattens this object and everything in it to a stream buffer.

11.9.3.14 void SILVERLINING_API SilverLining::LocalTime::SetDay (int *day*) [inline]

Sets the day of the month to simulate.

Parameters

day	The day of month, ranging from 1-31.

See also

GetDay()

11.9.3.15 void SILVERLINING_API SilverLining::LocalTime::SetFromEpochSeconds (time_t *time*)

Populates the LocalTime class based on epoch seconds (ie, as returned from `time(NULL)`).

Internally, it calls `gmtime()` on the value passed in, and populates the class based on the its results. The time zone will be set to GMT.

Parameters

time	Seconds elapsed since midnight, January 1, 1970 UTC.

11.9.3.16 void SILVERLINING_API SilverLining::LocalTime::SetFromSystemTime ( )

Populate the object based on the system's current local time settings.

11.9.3.17 void SILVERLINING_API SilverLining::LocalTime::SetHour (int *hour*) [inline]

Sets the hour of day to simulate.

Parameters

hour	The hour of day, ranging from 0-23 (0=midnight, 12=noon, 23=11PM)

See also

GetHour()

11.9.3.18 void SILVERLINING_API SilverLining::LocalTime::SetMinutes (int *minutes*) [inline]

Sets the minute of the hour to simulate.

Parameters

minutes	Minutes from 0-59

See also

GetMinutes()

11.9.3.19 void SILVERLINING_API SilverLining::LocalTime::SetMonth (int *month*) [inline]

Sets the calendar month to simulate.

Parameters

month	The month to simulate, ranging from 1 (January) to 12 (December).

See also

GetMonth();

11.9.3.20 void SILVERLINING_API SilverLining::LocalTime::SetObservingDaylightSavingsTime (bool *inDST*) [inline]

Sets if Daylight Savings Time is currently observed in the simulation.

Not required if using SetFromEpochSeconds(), but must be set otherwise.

See also

GetObservingDaylightSavingsTime()

11.9.3.21 void SILVERLINING_API SilverLining::LocalTime::SetSeconds (double *seconds*) [inline]

Sets the second of the minute to simulate.

Parameters

seconds	Seconds from 0-59.

See also

GetSeconds()

11.9.3.22 void SILVERLINING_API SilverLining::LocalTime::SetTimeZone (double *zone*) [inline]

Specifies the time zone being simulated.

This is an hour offset from GMT, use the TimeZones enumerated type as a convenient way to specify the hour offset for a known time zone. Be sure that this time zone is consistent with the Location you specify, unless you choose to specify all times in GMT consistently instead of using local time.

Parameters

zone	Hour offset from GMT, ignoring daylight savings time. ie, PST is -8. Use the TimeZones enumeration to obtain the hour offset for specific time zones.

See also

GetTimeZone()

11.9.3.23 void SILVERLINING_API SilverLining::LocalTime::SetYear (int *year*) [inline]

Set the calendar year to simulate.

SilverLining can only handle Gregorian calendar years, which means that years before 1582 will not be accurately simulated.

See also

GetYear()

The documentation for this class was generated from the following file:

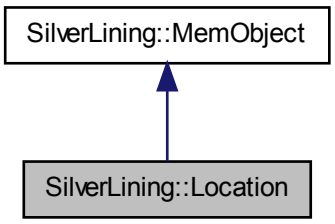
- C:/sl/Public Headers/LocalTime.h

11.10 SilverLining::Location Class Reference

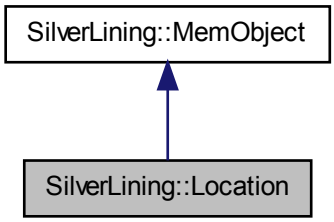
An object representing a geographic location.

`#include <Location.h>`

Inheritance diagram for SilverLining::Location:



Collaboration diagram for SilverLining::Location:



Public Member Functions

- Location ()
  Default constructor.
- void SILVERLINING_API SetLatitude (double decimalLatitude)
  Set the simulated latitude.
- double SILVERLINING_API GetLatitude () const
  Retrieves the simulated latitude in decimal degrees form.
- void SILVERLINING_API SetLongitude (double decimalLongitude)
  Set the simulated longitude.
- double SILVERLINING_API GetLongitude () const
  Retrieves the simulated longitude in decimal degrees.
- void SILVERLINING_API SetAltitude (double metersAboveMeanSeaLevel)
  Sets the simulated eyepoint altitude in meters above mean sea level.
- double SILVERLINING_API GetAltitude () const
  Retrives the currently simulated altitude above mean sea level, in meters.
- bool SILVERLINING_API Serialize (std::ostream &stream)
  Flattens this object and everything in it to a stream buffer.

- bool SILVERLINING_API Unserialize (std::istream &stream)

  Restores this object from the stream created using Serialize()

11.10.1 Detailed Description

An object representing a geographic location. A Location object must be passed to AtmosphericConditions::SetLocation() to specify the location you wish to simulate. This affects the position of the sun and moon in the sky, which in turn affects your scene's lighting. Be sure that your Location is consistent with the time zone specified in LocalTime::SetTimeZone(), unless you are choose to set all times as GMT/UTC rather than local time.

11.10.2 Constructor & Destructor Documentation

11.10.2.1 SilverLining::Location::Location ( )

Default constructor.

Creates a Location object with default settings. Defaults will be read from the SilverLining.config file if this is constructed after calling Atmosphere::Initialize()

11.10.3 Member Function Documentation

11.10.3.1 double SILVERLINING_API SilverLining::Location::GetAltitude ( ) const [inline]

Retrives the currently simulated altitude above mean sea level, in meters.

See also

SetAltitude()

11.10.3.2 double SILVERLINING_API SilverLining::Location::GetLatitude ( ) const [inline]

Retrieves the simulated latitude in decimal degrees form.

North is positive.

See also

SetLatitude()

11.10.3.3 double SILVERLINING_API SilverLining::Location::GetLongitude ( ) const [inline]

Retrieves the simulated longitude in decimal degrees.

East is positive.

See also

SetLongitude()

11.10.3.4 bool SILVERLINING_API SilverLining::Location::Serialize (std::ostream & *stream*)

Flattens this object and everything in it to a stream buffer.

11.10.3.5 void SILVERLINING_API SilverLining::Location::SetAltitude (double *metersAboveMeanSeaLevel*) [inline]

Sets the simulated eyepoint altitude in meters above mean sea level.

This altitude is only used for astronomical calculations and thereby doesn't have much of an affect on the simulation. Therefore it is not necessary to call SetAltitude() whenever your camera position changes, nor is it particularly important to set this to anything other than zero.

See also

GetAltitude()

11.10.3.6 void SILVERLINING_API SilverLining::Location::SetLatitude (double *decimalLatitude*) [inline]

Set the simulated latitude.

Parameters

decimalLatitude	The latitude in decimal degrees form (ie, 45.87). North is positive.

See also

GetLatitude()

11.10.3.7 void SILVERLINING_API SilverLining::Location::SetLongitude (double *decimalLongitude*) [inline]

Set the simulated longitude.

Parameters

decimalLongitude	The longitude in decimal degrees form (ie, -122.78). East is positive.

See also

GetLongitude()

The documentation for this class was generated from the following file:

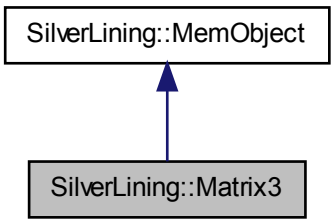
- C:/sl/Public Headers/Location.h

11.11 SilverLining::Matrix3 Class Reference

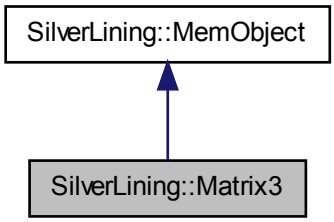
A simple 3x3 matrix class and its operations.

`#include <Matrix3.h>`

Inheritance diagram for SilverLining::Matrix3:



Collaboration diagram for SilverLining::Matrix3:



Public Member Functions

- Matrix3 ()

  Default contructor; performs no initialization for efficiency.

- Matrix3 (double e11, double e12, double e13, double e21, double e22, double e23, double e31, double e32, double e33)

  Constructor that instantiates the 3x3 matrix with initial values.

- ~Matrix3 ()

  Destructor.

- void SILVERLINING_API FromRx (double rad)

  Populates the matrix to model a rotation about the X axis by a given amount, in radians.

- void SILVERLINING_API FromRy (double rad)

  Populates the matrix to model a rotation about the Y axis by a given amount, in radians.

- void SILVERLINING_API FromRz (double rad)

  Populates the matrix to model a rotation about the Z axis by a given amount, in radians.

- void SILVERLINING_API FromXYZ (double Rx, double Ry, double Rz)

  Populates the matrix as a series of rotations about the X, Y, and Z axes (in that order) by specified amounts in radians.

- void SILVERLINING_API FromAxisAngle (const Vector3 &axis, double angle)

  Create a rotation matrix from axis and angle.

- Matrix3 SILVERLINING_API operator* (const Matrix3 &mat)

  Multiplies two matrices together.

- Vector3 SILVERLINING_API operator* (const Vector3 &rkVector) const

  Multiplies the matrix by a vector, yielding another 3x1 vector.

- Matrix3 SILVERLINING_API Transpose () const

  Calculate the transpose of the matrix.

- Matrix3 SILVERLINING_API Inverse () const

  Calculate the inverse of the matrix.

Public Attributes

- double elem [3][3]

  The data members are public for convenience.

Friends

- Vector3 SILVERLINING_API operator* (const Vector3 &vec, const Matrix3 &mat)

  Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

11.11.1 Detailed Description

A simple 3x3 matrix class and its operations.

11.11.2 Constructor & Destructor Documentation

11.11.2.1 SilverLining::Matrix3::Matrix3 ( ) [inline]

Default contructor; performs no initialization for efficiency.

11.11.2.2 SilverLining::Matrix3::Matrix3 (double *e11,* double *e12,* double *e13,* double *e21,* double *e22,* double *e23,* double *e31,* double *e32,* double *e33*) [inline]

Constructor that instantiates the 3x3 matrix with initial values.

11.11.2.3 SilverLining::Matrix3::~Matrix3 ( ) [inline]

Destructor.

11.11.3 Member Function Documentation

11.11.3.1 void SILVERLINING_API SilverLining::Matrix3::FromAxisAngle (const Vector3 & *axis,* double *angle*)

Create a rotation matrix from axis and angle.

11.11.3.2 void SILVERLINING_API SilverLining::Matrix3::FromRx (double *rad*)

Populates the matrix to model a rotation about the X axis by a given amount, in radians.

11.11.3.3 void SILVERLINING_API SilverLining::Matrix3::FromRy (double *rad*)

Populates the matrix to model a rotation about the Y axis by a given amount, in radians.

11.11.3.4 void SILVERLINING_API SilverLining::Matrix3::FromRz (double *rad*)

Populates the matrix to model a rotation about the Z axis by a given amount, in radians.

11.11.3.5 void SILVERLINING_API SilverLining::Matrix3::FromXYZ (double *Rx,* double *Ry,* double *Rz*)

Populates the matrix as a series of rotations about the X, Y, and Z axes (in that order) by specified amounts in radians.

11.11.3.6 Matrix3 SILVERLINING_API SilverLining::Matrix3::Inverse ( ) const

Calculate the inverse of the matrix.

11.11.3.7 Matrix3 SILVERLINING_API SilverLining::Matrix3::operator∗ (const Matrix3 & *mat*)

Multiplies two matrices together.

11.11.3.8 Vector3 SILVERLINING_API SilverLining::Matrix3::operator∗ (const Vector3 & *rkVector*) const

Multiplies the matrix by a vector, yielding another 3x1 vector.

11.11.3.9 Matrix3 SILVERLINING_API SilverLining::Matrix3::Transpose ( ) const

Calculate the transpose of the matrix.

11.11.4 Friends And Related Function Documentation

11.11.4.1 Vector3 SILVERLINING_API operator∗ (const Vector3 & *vec,* const Matrix3 & *mat*) [friend]

Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

The documentation for this class was generated from the following file:

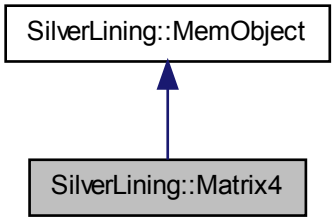
- C:/sl/Public Headers/Matrix3.h

11.12 SilverLining::Matrix4 Class Reference

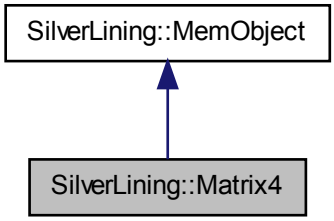
An implementation of a 4x4 matrix and some simple operations on it.

`#include <Matrix4.h>`

Inheritance diagram for SilverLining::Matrix4:



Collaboration diagram for SilverLining::Matrix4:



Public Member Functions

- Matrix4 ()

  Default constructor; initializes the matrix to an identity transform.

- Matrix4 (const Matrix3 &m3)

  Construct from a Matrix3.

- Matrix4 (double e11, double e12, double e13, double e14, double e21, double e22, double e23, double e24, double e31, double e32, double e33, double e34, double e41, double e42, double e43, double e44)

  This constructor allows you to initialize the matrix as you please.

- Matrix4 (double *m)

  Initializes the matrix from an array of 16 double-precision values (row-major).

- ~Matrix4 ()

  Destructor.

- Matrix4 SILVERLINING_API operator* (const Matrix4 &mat) const

  Multiplies two matrices together.

- Vector4 SILVERLINING_API operator* (const Vector4 &vec) const

  Transform a point by the matrix.

- Vector3 SILVERLINING_API operator* (const Vector3 &vec) const

  Transform a point by the matrix.

- void SILVERLINING_API Transpose ()

  Transposes the matrix in-place.

- double *SILVERLINING_API GetRow (int row)

  Retrieves a pointer into the requested row of the matrix.

Public Attributes

- double elem [4][4]

  Data members are public for convenience.

Friends

- Vector4 SILVERLINING_API operator* (const Vector4 &vec, const Matrix4 &mat)

  Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

11.12.1 Detailed Description

An implementation of a 4x4 matrix and some simple operations on it.

11.12.2 Constructor & Destructor Documentation

11.12.2.1 SilverLining::Matrix4::Matrix4 ( ) [inline]

Default constructor; initializes the matrix to an identity transform.

11.12.2.2 SilverLining::Matrix4::Matrix4 (const Matrix3 & *m3*) [inline]

Construct from a Matrix3.

11.12.2.3 SilverLining::Matrix4::Matrix4 (double *e11*, double *e12*, double *e13*, double *e14*, double *e21*, double *e22*, double *e23*, double *e24*, double *e31*, double *e32*, double *e33*, double *e34*, double *e41*, double *e42*, double *e43*, double *e44*) [inline]

This constructor allows you to initialize the matrix as you please.

11.12.2.4 SilverLining::Matrix4::Matrix4 (double * *m*) [inline]

Initializes the matrix from an array of 16 double-precision values (row-major).

11.12.2.5 SilverLining::Matrix4::~Matrix4 ( ) [inline]

Destructor.

11.12.3 Member Function Documentation

11.12.3.1 double* SILVERLINING_API SilverLining::Matrix4::GetRow (int *row*) [inline]

Retrieves a pointer into the requested row of the matrix.

11.12.3.2 Vector4 SILVERLINING_API SilverLining::Matrix4::operator* (const Vector4 & *vec*) const

Transform a point by the matrix.

11.12.3.3 Vector3 SILVERLINING_API SilverLining::Matrix4::operator∗ (const Vector3 & *vec*) const

Transform a point by the matrix.

11.12.3.4 Matrix4 SILVERLINING_API SilverLining::Matrix4::operator∗ (const Matrix4 & *mat*) const

Multiplies two matrices together.

11.12.3.5 void SILVERLINING_API SilverLining::Matrix4::Transpose ( ) [inline]

Transposes the matrix in-place.

11.12.4 Friends And Related Function Documentation

11.12.4.1 Vector4 SILVERLINING_API operator∗ (const Vector4 & *vec,* const Matrix4 & *mat*) [friend]

Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

The documentation for this class was generated from the following file:

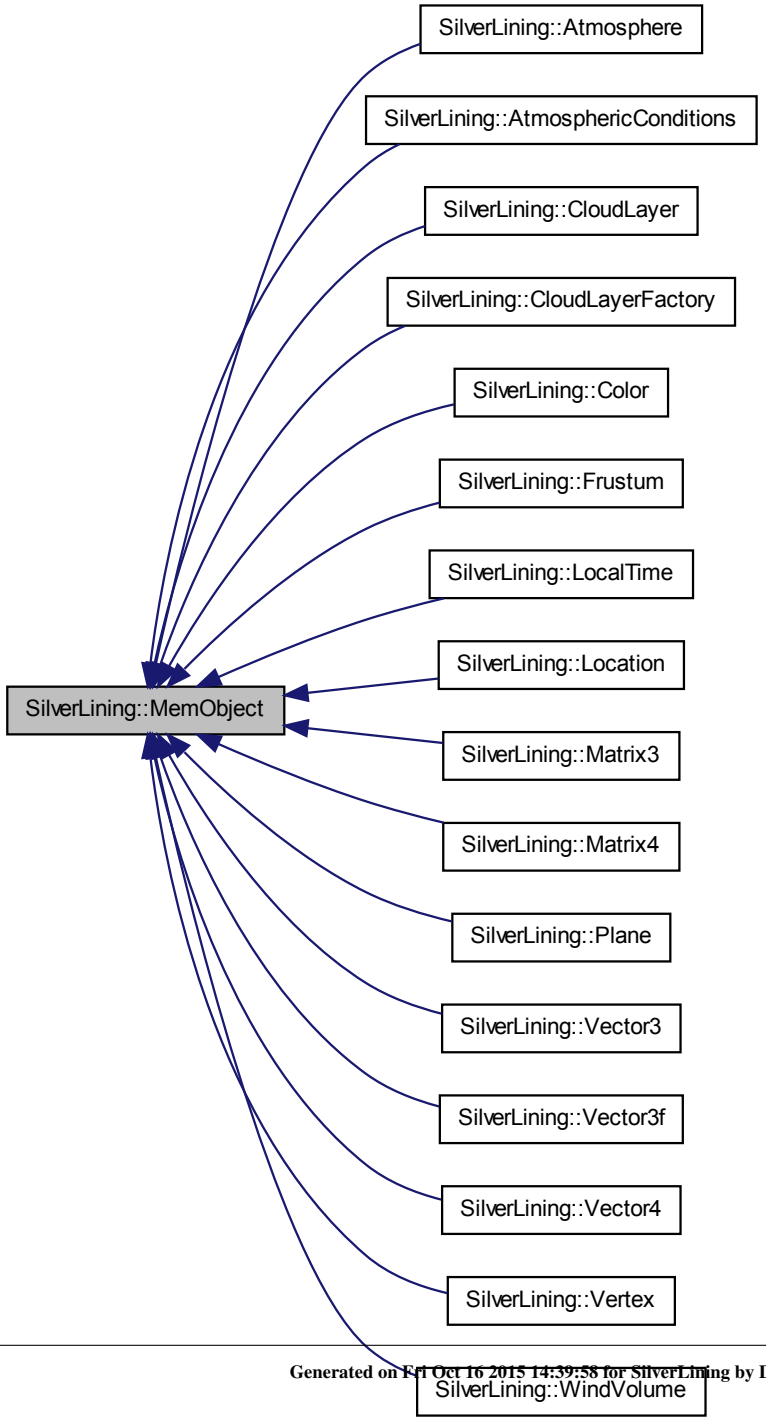
- C:/sl/Public Headers/Matrix4.h

11.13 SilverLining::MemObject Class Reference

This base class for all SilverLining objects intercepts the new and delete operators, routing them through SilverLining::Allocator().

`#include <MemAlloc.h>`

Inheritance diagram for SilverLining::MemObject:



11.13.1 Detailed Description

This base class for all SilverLining objects intercepts the new and delete operators, routing them through SilverLining::Allocator().

The documentation for this class was generated from the following file:

- C:/sl/Public Headers/MemAlloc.h

11.14 SilverLining::MillisecondTimer Class Reference

Exposes an interface to allow applications to control millisecond-level time.

`#include <MillisecondTimer.h>`

Public Member Functions

- MillisecondTimer ()

  Default constructor.

- virtual ~MillisecondTimer ()

  Virtual destructor.

- virtual unsigned long SILVERLINING_API GetMilliseconds () const

  Returns elapsed milliseconds.

11.14.1 Detailed Description

Exposes an interface to allow applications to control millisecond-level time. This timer is used to move cloud decks with respect to wind, and to simulate the real-time passage of time when AtmosphericConditions::EnableTimePassage() is activated.

By default, SilverLining will call the Win32 function timeGetTime() which will result in a simulation of real time that always moves forward. However, applications that want to move time backward for replays, or applications that want to simulate the acceleration of time, may pass in their own concept of elapsed milliseconds by extending this class.

To use your own MillisecondTimer in place of SilverLining's default implementation, instantiate your class that extends MillisecondTimer and implements its own GetMilliseconds() method. Then, pass this class instance to AtmosphericConditions::SetMillisecondTimer().

11.14.2 Constructor & Destructor Documentation

11.14.2.1 SilverLining::MillisecondTimer::MillisecondTimer ( ) [inline]

Default constructor.

11.14.2.2 virtual SilverLining::MillisecondTimer::~MillisecondTimer ( ) [inline, virtual]

Virtual destructor.

11.14.3 Member Function Documentation

11.14.3.1 virtual unsigned long SILVERLINING_API SilverLining::MillisecondTimer::GetMilliseconds ( ) const [virtual]

Returns elapsed milliseconds.

It doesn't matter what time this is relative to; SilverLining will only use the difference between subsequent calls to GetMilliseconds() to simulate the movement of the clouds and astronomical objects over time. The base class implementation of GetMilliseconds() simply calls the system function timeGetTime().

Avoid returning a value of zero; this value is special internally to flag the first frame.

The documentation for this class was generated from the following file:

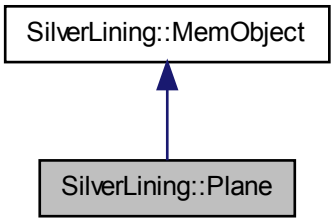
- C:/sl/Public Headers/MillisecondTimer.h

11.15 SilverLining::Plane Class Reference

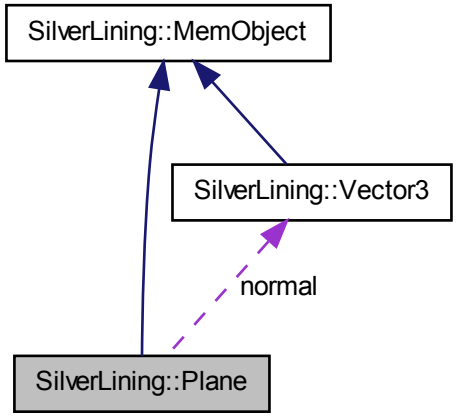
A geometric plane, modelled as a normal vector and a distance to the plane.

`#include <Plane.h>`

Inheritance diagram for SilverLining::Plane:



Collaboration diagram for SilverLining::Plane:



Public Member Functions

- Plane ()

Default constructor, performs no intializations.

- Plane (double a, double b, double c, double d)

  Constructor that initializes the plane based on coefficients of the plane equation, ax + by + cz + d = 0.

- Plane (Vector3 pNormal, double pDistance)

  Constructor that initializes the plane based on a normal vector and a distance to the plane, which is how the plane is represented internally.

- const Vector3 &SILVERLINING_API GetNormal () const

  Retrieves the normal vector for the plane.

- double SILVERLINING_API GetDistance () const

  Retrieves the distance to the plane.

- void SILVERLINING_API SetNormal (const Vector3 &v)

  Set the normal vector for the plane.

- void SILVERLINING_API SetDistance (double d)

  Set the distance to the plane.

- void SILVERLINING_API Normalize ()

  Normalizes the plane normal vector and adjusts the distance accordingly.

11.15.1 Detailed Description

A geometric plane, modelled as a normal vector and a distance to the plane.

11.15.2 Constructor & Destructor Documentation

11.15.2.1 SilverLining::Plane::Plane (double *a*, double *b*, double *c*, double *d*) [inline]

Constructor that initializes the plane based on coefficients of the plane equation, ax + by + cz + d = 0.

11.15.2.2 SilverLining::Plane::Plane (Vector3 *pNormal*, double *pDistance*) [inline]

Constructor that initializes the plane based on a normal vector and a distance to the plane, which is how the plane is represented internally.

11.15.3 Member Function Documentation

11.15.3.1 double SILVERLINING_API SilverLining::Plane::GetDistance ( ) const [inline]

Retrieves the distance to the plane.

11.15.3.2 const Vector3& SILVERLINING_API SilverLining::Plane::GetNormal ( ) const [inline]

Retrieves the normal vector for the plane.

11.15.3.3 void SILVERLINING_API SilverLining::Plane::Normalize ( ) [inline]

Normalizes the plane normal vector and adjusts the distance accordingly.

11.15.3.4 void SILVERLINING_API SilverLining::Plane::SetDistance (double *d*) [inline]

Set the distance to the plane.

11.15.3.5 void SILVERLINING_API SilverLining::Plane::SetNormal (const Vector3 & *v*) [inline]

Set the normal vector for the plane.

The documentation for this class was generated from the following file:

- C:/sl/Public Headers/Plane.h

11.16 SilverLining::RandomNumberGenerator Class Reference

Extend this class in order to use your own random number generator with SilverLining.

`#include <RandomNumberGenerator.h>`

Public Member Functions

- virtual ~RandomNumberGenerator ()

  Virtual destructor.

- virtual float SILVERLINING_API UniformRandomFloat () const =0

  Return a uniformly distributed random float between 0.0 and 1.0.

- virtual double SILVERLINING_API UniformRandomDouble () const =0

  Return a uniformly distributed random double between 0.0 and 1.0.

- virtual unsigned int SILVERLINING_API UniformRandomIntRange (unsigned int start, unsigned int end) const =0

  Return a uniformly distributed random integer between the integers specified (inclusive.)

11.16.1 Detailed Description

Extend this class in order to use your own random number generator with SilverLining. Pass your implementation into Atmosphere::SetRandomNumberGenerator().

11.16.2 Constructor & Destructor Documentation

11.16.2.1 virtual SilverLining::RandomNumberGenerator::~RandomNumberGenerator ( )

`[inline, virtual]`

Virtual destructor.

The documentation for this class was generated from the following file:

- C:/sl/Public Headers/RandomNumberGenerator.h

11.17 SilverLining::ResourceLoader Class Reference

This class is used whenever SilverLining needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish.

```
#include <ResourceLoader.h>
```

Public Member Functions

- ResourceLoader ()

  Constructor.

- virtual ~ResourceLoader ()

  Virtual destructor; frees the memory of the resource path string.

- void SILVERLINING_API SetResourceDirPath (const char *path)

  Sets the path to the SilverLining resources folder, which will be pre-pended to all resource filenames passed into LoadResource().

- const char *SILVERLINING_API GetResourceDirPath () const

  Retrieves the path set by SetResourceDirPath().

- virtual bool SILVERLINING_API LoadResource (const char *pathName, char *&data, unsigned int &dataLen, bool text)

  Load a resource from mass storage; the default implementation uses the POSIX functions fopen(), fread(), and fclose() to do this, but you may override this method to load resources however you wish.

- virtual void SILVERLINING_API FreeResource (char *data)

  Frees the resource data memory that was returned from LoadResource().

- virtual bool SILVERLINING_API GetFilesInDirectory (const char *pathName, SL_VECTOR(SL_STRING)&dirContents)

  Retrieves a list of file names within the directory path specified (relative to the resource path specified with SetResourceDirPath().

11.17.1 Detailed Description

This class is used whenever SilverLining needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish. If you have your own system of packed files, you can include SilverLining's resources directory into it and implement your own ResourceLoader to access our resources within your pack files.

Custom ResourceLoader classes should be passed into Atmosphere::SetResourceLoader() after constructing the Atmosphere class. The Atmosphere maintains a static instance of ResourceLoader that is shared across all Atmosphere objects.

If you choose to implement your own resource management scheme, you'll need to take care with the rendering DLL's found under the VC6, VC7, VC8, and VC9 directories of the SilverLining resource folder. Windows can only load DLL's directly from a file, so you'll need to exclude these DLL's from your pack files and leave them loose in the resources directory. Alternately, you may eliminate the DLL's by statically linking the renderer you want into the SilverLining libraries themselves. See the "Overview of SilverLining" section of the documentation for more information.

11.17.2 Member Function Documentation

11.17.2.1 virtual void SILVERLINING_API SilverLining::ResourceLoader::FreeResource (char * *data*) [virtual]

Frees the resource data memory that was returned from LoadResource().

The data pointer will be invalid following this call.

11.17.2.2 virtual bool SILVERLINING_API SilverLining::ResourceLoader::GetFilesInDirectory (const char * *pathName*, SL_VECTOR(SL_STRING)& *dirContents*) [virtual]

Retrieves a list of file names within the directory path specified (relative to the resource path specified with SetResourceDirPath().

Parameters

pathName	The path to the directory underneath the resources directory. The path to the resources directory will be pre-pended to this path.
dirContents	A reference that will receive a vector of strings of the file names found inside this path, if any.

Returns

True if the path was found and scanned successfully, false otherwise.

11.17.2.3 virtual bool SILVERLINING_API SilverLining::ResourceLoader::LoadResource (const char * *pathName*, char *& *data*, unsigned int & *dataLen*, bool *text*) [virtual]

Load a resource from mass storage; the default implementation uses the POSIX functions fopen(), fread(), and fclose() to do this, but you may override this method to load resources however you wish.

The caller is responsible for calling FreeResource() when it's done consuming the resource data in order to free its memory.

Parameters

pathName	The path to the desired resource, relative to the location of the resources folder previously specified in SetResourceDirPath().
data	A reference to a char * that will return the resource's data upon a successful load.
dataLen	A reference to an unsigned int that will return the number of bytes loaded upon a successful load.
text	True if the resource is a text file, such as a shader. If true, a terminating null character will be appended to the resulting data and the file will be opened in text mode.

Returns

True if the resource was located and loaded successfully, false otherwise.

See also

SetResourceDirPath

11.17.2.4 void SILVERLINING_API SilverLining::ResourceLoader::SetResourceDirPath (const char * *path*)

Sets the path to the SilverLining resources folder, which will be pre-pended to all resource filenames passed into LoadResource().

This path is also used to locate the renderer DLL's inside the SilverLining resources folder. It should be called after constructing the ResourceLoader and before calling LoadResource().

The documentation for this class was generated from the following file:

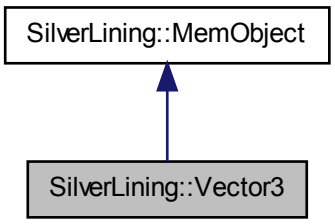
- C:/sl/Public Headers/ResourceLoader.h

11.18 SilverLining::Vector3 Class Reference

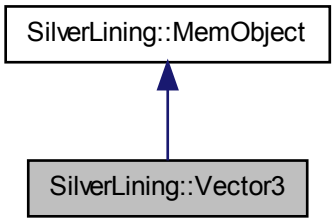
A 3D double-precision Vector class and its operations.

`#include <Vector3.h>`

Inheritance diagram for SilverLining::Vector3:



Collaboration diagram for SilverLining::Vector3:



Public Member Functions

- Vector3 ()

  Default constructor, initializes to (0,0,0).

- Vector3 (double px, double py, double pz)

  Constructs a Vector3 with the given x,y,z coordinates.

- Vector3 (const Vector3 &v)

  Copy constructor.

- double SILVERLINING_API Length () const

  Returns the length of the vector.

- double SILVERLINING_API SquaredLength () const

  Returns the squared length of the vector, which is faster than computing the length.

- void SILVERLINING_API Normalize ()

  Scales the vector to be of length 1.0.

- double SILVERLINING_API Dot (const Vector3 &v) const

  Determines the dot product between this vector and another, and returns the result.

- Vector3 SILVERLINING_API Cross (const Vector3 &v) const

  Determines the cross product between this vector and another, and returns the result.

- Vector3 SILVERLINING_API operator∗ (double n) const

  Scales each x,y,z value of the vector by a constant n, and returns the result.

- Vector3 SILVERLINING_API operator+ (double n) const

  Adds a constant n to each component of the vector, and returns the result.

- Vector3 SILVERLINING_API operator- (const Vector3 &v) const

  Subtracts the specified vector from this vector, and returns the result.

- Vector3 SILVERLINING_API operator+ (const Vector3 &v) const

  Adds this vector to the specified vector, and returns the result.

- Vector3 SILVERLINING_API operator∗ (const Vector3 &v) const

  Multiply two vectors together.

- bool SILVERLINING_API operator== (const Vector3 &v) const

  Tests if two vectors are exactly equal.

- bool SILVERLINING_API operator!= (const Vector3 &v) const

  Test if two vectors are not exactly equal.

- void SILVERLINING_API Serialize (std::ostream &s) const

  Write this vector's data to a file.

- void SILVERLINING_API Unserialize (std::istream &s)

  Restore this vector from a file.

Public Attributes

- double x

  Data members x,y,z public for convenience.

11.18.1 Detailed Description

A 3D double-precision Vector class and its operations.

11.18.2 Constructor & Destructor Documentation

11.18.2.1 SilverLining::Vector3::Vector3 ( ) [inline]

Default constructor, initializes to (0,0,0).

11.18.2.2 SilverLining::Vector3::Vector3 (double *px*, double *py*, double *pz*) [inline]

Constructs a Vector3 with the given x,y,z coordinates.

11.18.3 Member Function Documentation

11.18.3.1 Vector3 SILVERLINING_API SilverLining::Vector3::Cross (const Vector3 & *v*) const [inline]

Determines the cross product between this vector and another, and returns the result.

11.18.3.2 double SILVERLINING_API SilverLining::Vector3::Dot (const Vector3 & *v*) const [inline]

Determines the dot product between this vector and another, and returns the result.

11.18.3.3 double SILVERLINING_API SilverLining::Vector3::Length ( ) const [inline]

Returns the length of the vector.

11.18.3.4 void SILVERLINING_API SilverLining::Vector3::Normalize ( ) [inline]

Scales the vector to be of length 1.0.

11.18.3.5 bool SILVERLINING_API SilverLining::Vector3::operator!= (const Vector3 & *v*) const [inline]

Test if two vectors are not exactly equal.

11.18.3.6 Vector3 SILVERLINING_API SilverLining::Vector3::operator* (const Vector3 & *v*) const [inline]

Multiply two vectors together.

11.18.3.7 Vector3 SILVERLINING_API SilverLining::Vector3::operator* (double *n*) const [inline]

Scales each x,y,z value of the vector by a constant n, and returns the result.

11.18.3.8 Vector3 SILVERLINING_API SilverLining::Vector3::operator+ (const Vector3 & *v*) const [inline]

Adds this vector to the specified vector, and returns the result.

11.18.3.9 Vector3 SILVERLINING_API SilverLining::Vector3::operator+ (double *n*) const [inline]

Adds a constant n to each component of the vector, and returns the result.

11.18.3.10 Vector3 SILVERLINING_API SilverLining::Vector3::operator- (const Vector3 & *v*) const [inline]

Subtracts the specified vector from this vector, and returns the result.

11.18.3.11 bool SILVERLINING_API SilverLining::Vector3::operator== (const Vector3 & *v*) const [inline]

Tests if two vectors are exactly equal.

11.18.3.12 void SILVERLINING_API SilverLining::Vector3::Serialize (std::ostream & *s*) const [inline]

Write this vector's data to a file.

11.18.3.13 double SILVERLINING_API SilverLining::Vector3::SquaredLength ( ) const [inline]

Returns the squared length of the vector, which is faster than computing the length.

11.18.3.14 void SILVERLINING_API SilverLining::Vector3::Unserialize (std::istream & *s*) [inline]

Restore this vector from a file.

11.18.4 Member Data Documentation

11.18.4.1 double SilverLining::Vector3::x

Data members x,y,z public for convenience.

The documentation for this class was generated from the following file:

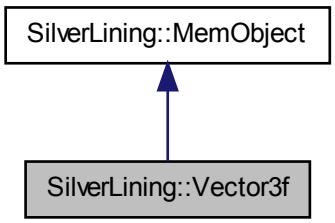
- C:/sl/Public Headers/Vector3.h

11.19 SilverLining::Vector3f Class Reference

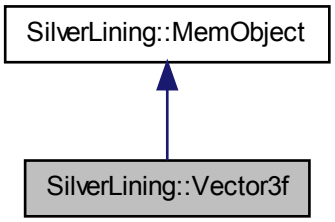
A 3D single-precision vector class, and its operations.

`#include <Vector3.h>`

Inheritance diagram for SilverLining::Vector3f:



Collaboration diagram for SilverLining::Vector3f:



Public Member Functions

- Vector3f ()

  Default constructor; does not initialize the vector.

- Vector3f (const SilverLining::Vector3 &v)

  Construct a single precision vector from a double precision one.

- Vector3f (float px, float py, float pz)

  Constructs a Vector3f from the specified single-precision floating point x, y, and z values.

- float SILVERLINING_API Length ()

  Returns the length of this vector.

- void SILVERLINING_API Normalize ()

  Scales the vector to be of length 1.0.

- double SILVERLINING_API Dot (const Vector3 &v) const

  Returns the dot product of this vector with the specified Vector3.

- Vector3f SILVERLINING_API operator- (const Vector3f &v) const

  Subtracts the specified vector from this vector, and returns the result.

- Vector3f SILVERLINING_API operator+ (const Vector3f &v) const

  Adds this vector to the specified vector, and returns the result.

- Vector3f SILVERLINING_API operator∗ (float s) const

  Multiplies this vector by a scalar and returns the result.

Public Attributes

- float x

  Data members x, y, z are public for convenience.

11.19.1 Detailed Description

A 3D single-precision vector class, and its operations.

11.19.2 Constructor & Destructor Documentation

11.19.2.1 SilverLining::Vector3f::Vector3f ( ) [inline]

Default constructor; does not initialize the vector.

11.19.2.2 SilverLining::Vector3f::Vector3f (const SilverLining::Vector3 & *v*) [inline]

Construct a single precision vector from a double precision one.

11.19.2.3 SilverLining::Vector3f::Vector3f (float *px*, float *py*, float *pz*) [inline]

Constructs a Vector3f from the specified single-precision floating point x, y, and z values.

11.19.3 Member Function Documentation

11.19.3.1 double SILVERLINING_API SilverLining::Vector3f::Dot (const Vector3 & *v*) const [inline]

Returns the dot product of this vector with the specified Vector3.

11.19.3.2 float SILVERLINING_API SilverLining::Vector3f::Length ( ) [inline]

Returns the length of this vector.

11.19.3.3 void SILVERLINING_API SilverLining::Vector3f::Normalize ( ) [inline]

Scales the vector to be of length 1.0.

11.19.3.4 Vector3f SILVERLINING_API SilverLining::Vector3f::operator* (float *s*) const [inline]

Multiplies this vector by a scalar and returns the result.

11.19.3.5 Vector3f SILVERLINING_API SilverLining::Vector3f::operator+ (const Vector3f & *v*) const [inline]

Adds this vector to the specified vector, and returns the result.

11.19.3.6 Vector3f SILVERLINING_API SilverLining::Vector3f::operator- (const Vector3f & *v*) const [inline]

Subtracts the specified vector from this vector, and returns the result.

11.19.4 Member Data Documentation

11.19.4.1 float SilverLining::Vector3f::x

Data members x, y, z are public for convenience.

The documentation for this class was generated from the following file:

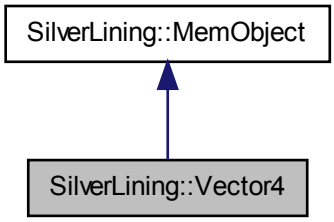
- C:/sl/Public Headers/Vector3.h

11.20 SilverLining::Vector4 Class Reference

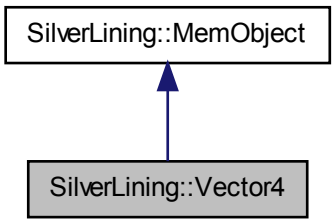
A simple double-precision 4D vector class with no operations defined.

`#include <Vector4.h>`

Inheritance diagram for SilverLining::Vector4:



Collaboration diagram for SilverLining::Vector4:



Public Member Functions

- Vector4 (double px, double py, double pz, double pw)

  Constructs a Vector4 from the given x, y, z, and w values.

- Vector4 (const Vector3 &v3)

  Constructs a Vector4 from a Vector3, setting w to 1.

- Vector4 ()

Default constructor; initializes the Vector4 to (0, 0, 0, 1)

- double SILVERLINING_API Dot (const Vector4 &v) const

  Determines the dot product between this vector and another, and returns the result.

- Vector4 SILVERLINING_API operator* (double n) const

  Scales each x,y,z value of the vector by a constant n, and returns the result.

Public Attributes

- double x

  The x, y, z, and w data members are public for convenience.

11.20.1 Detailed Description

A simple double-precision 4D vector class with no operations defined. Essentially a struct with constructors.

11.20.2 Constructor & Destructor Documentation

11.20.2.1 SilverLining::Vector4::Vector4 (double *px*, double *py*, double *pz*, double *pw*) [inline]

Constructs a Vector4 from the given x, y, z, and w values.

11.20.2.2 SilverLining::Vector4::Vector4 (const Vector3 & *v3*) [inline]

Constructs a Vector4 from a Vector3, setting w to 1.

11.20.3 Member Function Documentation

11.20.3.1 double SILVERLINING_API SilverLining::Vector4::Dot (const Vector4 & *v*) const [inline]

Determines the dot product between this vector and another, and returns the result.

11.20.3.2 Vector4 SILVERLINING_API SilverLining::Vector4::operator* (double *n*) const [inline]

Scales each x,y,z value of the vector by a constant n, and returns the result.

11.20.4 Member Data Documentation

11.20.4.1 double SilverLining::Vector4::x

The x, y, z, and w data members are public for convenience.

The documentation for this class was generated from the following file:

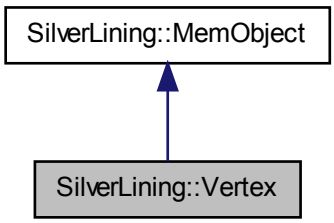
- C:/sl/Public Headers/Vector4.h

11.21 SilverLining::Vertex Class Reference

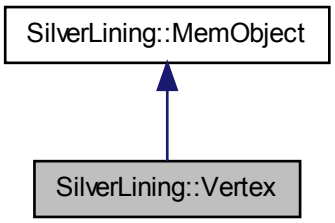
A single vertex containing a position, RGBA color, and 3D texture coordinates.

`#include <Vertex.h>`

Inheritance diagram for SilverLining::Vertex:



Collaboration diagram for SilverLining::Vertex:



Public Member Functions

- void SILVERLINING_API SetColor (unsigned char r, unsigned char g, unsigned char b, unsigned char a)

  Sets the 4-byte color of the vertex in the manner expected by the underlying Renderer, based on the 0-255 RGBA values provided.

- void SILVERLINING_API SetUV (float u, float v)

  Sets the u,v texture coordinates of the vertex in the manner expected by the underlying Renderer, based on the uv coordinates provided.

- void SILVERLINING_API SetColor (const Color &c)

  Sets the 4-byte color of the vertex in the manner expected by the underlying Renderer, based on the Color provided.

11.21.1 Detailed Description

A single vertex containing a position, RGBA color, and 3D texture coordinates.

11.21.2 Member Function Documentation

11.21.2.1 void SILVERLINING_API SilverLining::Vertex::SetColor (unsigned char *r,* unsigned char *g,* unsigned char *b,* unsigned char *a*)

Sets the 4-byte color of the vertex in the manner expected by the underlying Renderer, based on the 0-255 RGBA values provided.

11.21.2.2 void SILVERLINING_API SilverLining::Vertex::SetColor (const Color & *c*)

Sets the 4-byte color of the vertex in the manner expected by the underlying Renderer, based on the Color provided.

11.21.2.3 void SILVERLINING_API SilverLining::Vertex::SetUV (float *u,* float *v*)

Sets the u,v texture coordinates of the vertex in the manner expected by the underlying Renderer, based on the uv coordinates provided.

The r texture coordinate is set to 0.

The documentation for this class was generated from the following file:

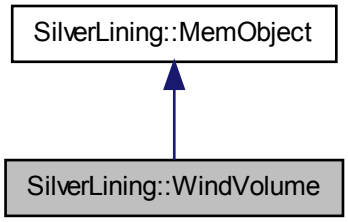
- C:/sl/Public Headers/Vertex.h

11.22 SilverLining::WindVolume Class Reference

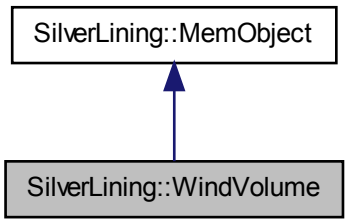
Defines an area of a given wind velocity and direction bounded by two altitudes.

`#include <WindVolume.h>`

Inheritance diagram for SilverLining::WindVolume:



Collaboration diagram for SilverLining::WindVolume:



Public Member Functions

- WindVolume ()

  Default constructor.

- virtual ~WindVolume ()

  Destructor.

- void SILVERLINING_API SetMinAltitude (double metersMSL)

Set the minimum altitude affected by this object's wind settings.

- double SILVERLINING_API GetMinAltitude () const

  Retrieves the minimum altitude, in meters above mean sea level, affected by this object.

- void SILVERLINING_API SetMaxAltitude (double metersMSL)

  Set the maximum altitude affected by this object's wind settings.

- double SILVERLINING_API GetMaxAltitude () const

  Retrieves the maximum altitude, in meters above mean sea level, affected by this object.

- void SILVERLINING_API SetWindSpeed (double metersPerSecond)

  Set the wind velocity within this WindVolume, in meters per second.

- double SILVERLINING_API GetWindSpeed () const

  Retrieves the wind velocity within this WindVolume, in meters per second.

- void SILVERLINING_API SetDirection (double degreesFromNorth)

  Sets the wind direction, in degrees East from North.

- double SILVERLINING_API GetDirection () const

  Retrieves the wind direction, in degrees East from North.

- bool SILVERLINING_API Inside (double metersMSL) const

  Evaluates if a given altitude is affected by this WindVolume.

- bool SILVERLINING_API Serialize (std::ostream &stream)

  Flattens this object and everything in it to a stream buffer.

- bool SILVERLINING_API Unserialize (std::istream &stream)

  Restores this object from the stream created using Serialize()

11.22.1 Detailed Description

Defines an area of a given wind velocity and direction bounded by two altitudes. Passed into AtmosphericConditions::SetWind() to define wind that will affect cloud motion.

11.22.2 Constructor & Destructor Documentation

11.22.2.1 SilverLining::WindVolume::WindVolume ( ) [inline]

Default constructor.

Creates a WindVolume with default settings of no wind from zero to 100,000 meters above mean sea level.

11.22.2.2 virtual SilverLining::WindVolume::~WindVolume ( ) [inline, virtual]

Destructor.

11.22.3 Member Function Documentation

11.22.3.1 double SILVERLINING_API SilverLining::WindVolume::GetDirection ( ) const [inline]

Retrieves the wind direction, in degrees East from North.

This is the direction the wind is blowing toward, not the direction it is coming from.

See also

SetDirection()

11.22.3.2 double SILVERLINING_API SilverLining::WindVolume::GetMaxAltitude ( ) const [inline]

Retrieves the maximum altitude, in meters above mean sea level, affected by this object.

See also

SetMaxAltitude()

11.22.3.3 double SILVERLINING_API SilverLining::WindVolume::GetMinAltitude ( ) const [inline]

Retrieves the minimum altitude, in meters above mean sea level, affected by this object.

See also

SetMinAltitude()

11.22.3.4 double SILVERLINING_API SilverLining::WindVolume::GetWindSpeed ( ) const [inline]

Retrieves the wind velocity within this WindVolume, in meters per second.

See also

SetWindSpeed()

11.22.3.5 bool SILVERLINING_API SilverLining::WindVolume::Inside (double *metersMSL*) const [inline]

Evaluates if a given altitude is affected by this WindVolume.

Parameters

metersMSL	The altitude to query on, in meters above mean sea level.

Returns

True if the given altitude is affected by this WindVolume object.

11.22.3.6 bool SILVERLINING_API SilverLining::WindVolume::Serialize (std::ostream & *stream*) [inline]

Flattens this object and everything in it to a stream buffer.

11.22.3.7 void SILVERLINING_API SilverLining::WindVolume::SetDirection (double *degreesFromNorth*) [inline]

Sets the wind direction, in degrees East from North.

This is the direction the wind is coming from, not the direction it is blowing toward.

See also

GetDirection()

11.22.3.8 void SILVERLINING_API SilverLining::WindVolume::SetMaxAltitude (double *metersMSL*) [inline]

Set the maximum altitude affected by this object's wind settings.

Parameters

metersMSL	The maximum altitude of this WindVolume in meters above mean sea level.

See also

GetMaxAltitude()

11.22.3.9 void SILVERLINING_API SilverLining::WindVolume::SetMinAltitude (double *metersMSL*) [inline]

Set the minimum altitude affected by this object's wind settings.

Parameters

metersMSL	The minimum altitude of this WindVolume in meters above mean sea level.

See also

GetMinAltitude()

11.22.3.10 void SILVERLINING_API SilverLining::WindVolume::SetWindSpeed (double *metersPerSecond*) [inline]

Set the wind velocity within this WindVolume, in meters per second.

See also

GetWindSpeed()

The documentation for this class was generated from the following file:

- C:/sl/Public Headers/WindVolume.h

Chapter 12

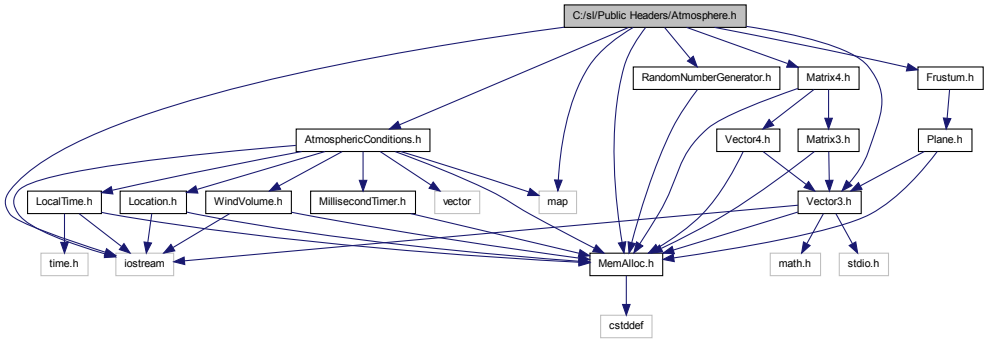
File Documentation

12.1 C:/sl/Public Headers/Atmosphere.h File Reference

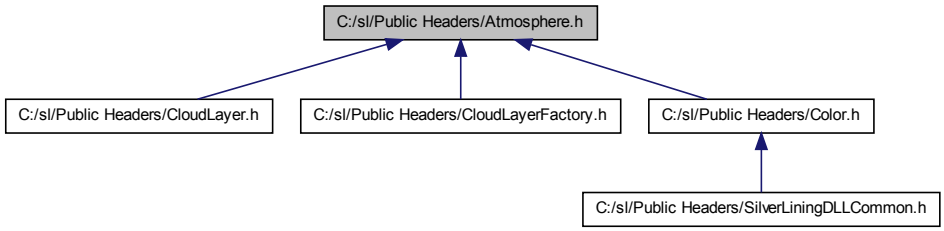
The main interface to SilverLining.

```
#include "SilverLiningTypes.h"
#include "MemAlloc.h"
#include "AtmosphericConditions.h"
#include "RandomNumberGenerator.h"
#include "Matrix4.h"
#include "Vector3.h"
#include "Frustum.h"
#include <map>
#include <iostream>
```

Include dependency graph for Atmosphere.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Atmosphere

  This class is the main interface to SilverLining.

12.1.1 Detailed Description

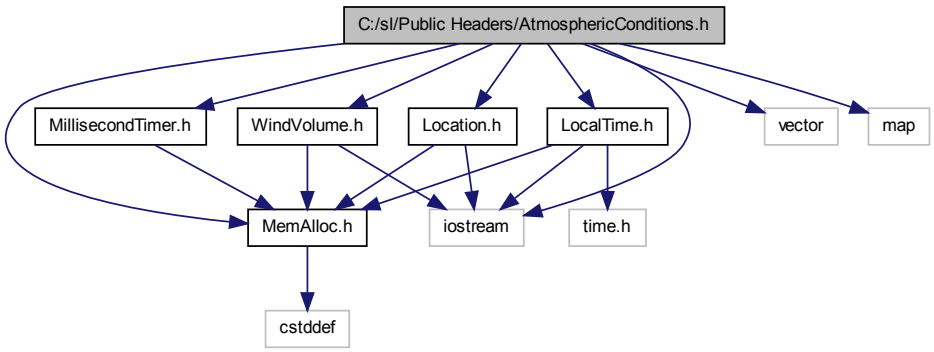
The main interface to SilverLining.

12.2 C:/sl/Public Headers/AtmosphericConditions.h File Reference

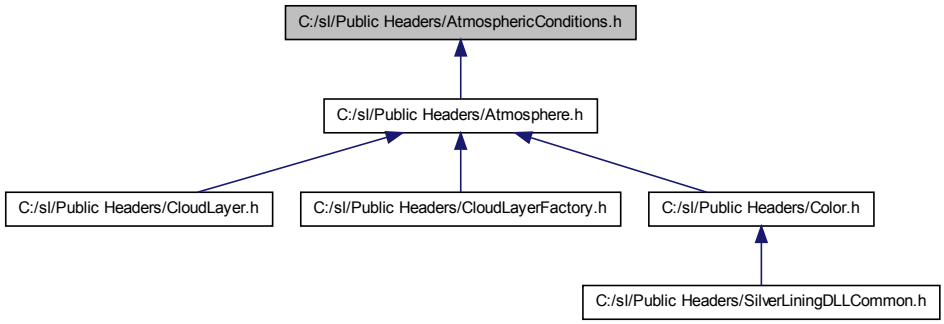
Configures the time, location, atmosphere, and cloud cover.

```
#include "MemAlloc.h"
#include "WindVolume.h"
#include "Location.h"
#include "LocalTime.h"
#include "MillisecondTimer.h"
#include <vector>
#include <map>
#include <iostream>
```

Include dependency graph for AtmosphericConditions.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::AtmosphericConditions

  A class to set the current simulation conditions for the sky.

12.2.1 Detailed Description

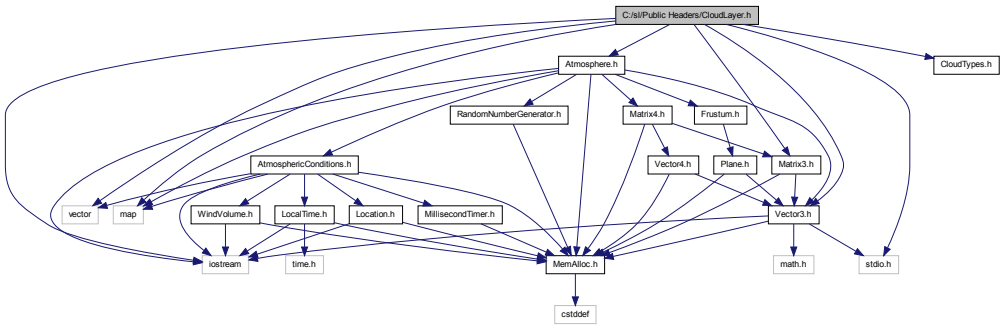
Configures the time, location, atmosphere, and cloud cover.

12.3 C:/sl/Public Headers/CloudLayer.h File Reference

Methods for configuration and initializing cloud decks.

```
#include "Atmosphere.h"
#include "Vector3.h"
#include "Matrix3.h"
#include "CloudTypes.h"
#include <vector>
#include <map>
#include <stdio.h>
#include <iostream>
```

Include dependency graph for CloudLayer.h:



Classes

- class SilverLining::CloudLayer

  A collection of clouds of uniform classification organized into a "deck" of similar altitudes.

12.3.1 Detailed Description

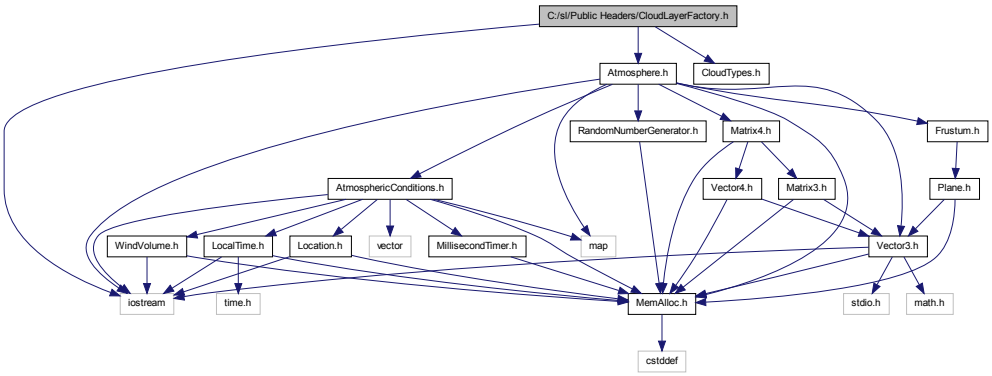
Methods for configuration and initializing cloud decks.

12.4 C:/sl/Public Headers/CloudLayerFactory.h File Reference

Instantiates specific types of CloudLayer objects.

```
#include "Atmosphere.h"
#include "CloudTypes.h"
#include <iostream>
```

Include dependency graph for CloudLayerFactory.h:



Classes

- class SilverLining::CloudLayerFactory

  A class factory for CloudLayer objects.

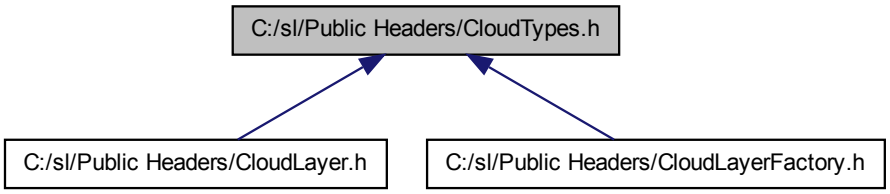
12.4.1 Detailed Description

Instantiates specific types of CloudLayer objects.

12.5 C:/sl/Public Headers/CloudTypes.h File Reference

Defines the enumerated values of cloud layer types that may be passed to CloudLayer-Factory::Create.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum CloudTypes {
  CIRROCUMULUS, CIRRUS_FIBRATUS, STRATUS,
  CUMULUS_MEDIOCRIS, CUMULUS_CONGESTUS, CUMULUS_CONGESTUS_-
  HI_RES,
  CUMULONIMBUS_CAPPILATUS, STRATOCUMULUS, TOWERING_CUMULUS,
  SANDSTORM, NUM_CLOUD_TYPES }

12.5.1 Detailed Description

Defines the enumerated values of cloud layer types that may be passed to CloudLayer-Factory::Create.

12.5.2 Enumeration Type Documentation

12.5.2.1 enum CloudTypes

Cloud types suitable for passing into CloudLayerFactory::Create

Enumerator:

CIRROCUMULUS High planar cloud puffs

CIRRUS_FIBRATUS High, thicker and fibrous clouds that signal changing weather

STRATUS Low clouds represented as a slab

CUMULUS_MEDIOCRIS Low, puffy clouds on fair days

CUMULUS_CONGESTUS Large cumulus clouds, built for performance.

CUMULUS_CONGESTUS_HI_RES Large cumulus clouds represented with high-resolution puff textures.

CUMULONIMBUS_CAPPILATUS Big storm clouds.

STRATOCUMULUS Low, dense, puffy clouds with some sun breaks between them.

TOWERING_CUMULUS Very large, tall cumulus clouds in the process of becoming a thunderstorm.

SANDSTORM A "haboob" cloud of dust intended to be positioned at ground level.

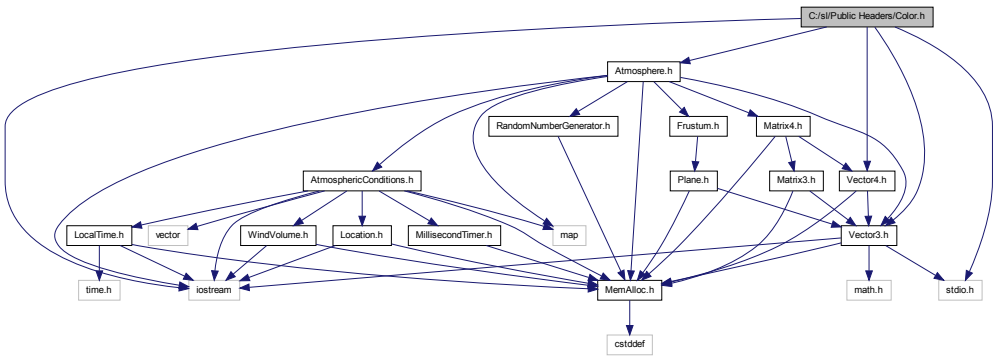
NUM_CLOUD_TYPES Total number of cloud types.

12.6 C:/sl/Public Headers/Color.h File Reference

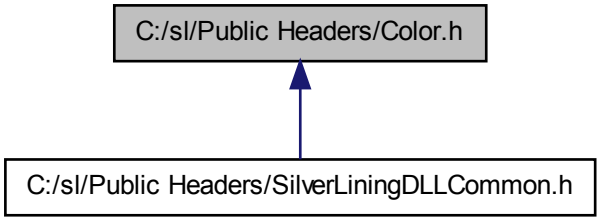
A class that defines an RGBA color and operations on it.

```
#include "Atmosphere.h"
#include "Vector3.h"
#include "Vector4.h"
#include <stdio.h>
#include <iostream>
```

Include dependency graph for Color.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Color

  A RGBA color, where each component is represented as a float from 0-1.

12.6.1 Detailed Description

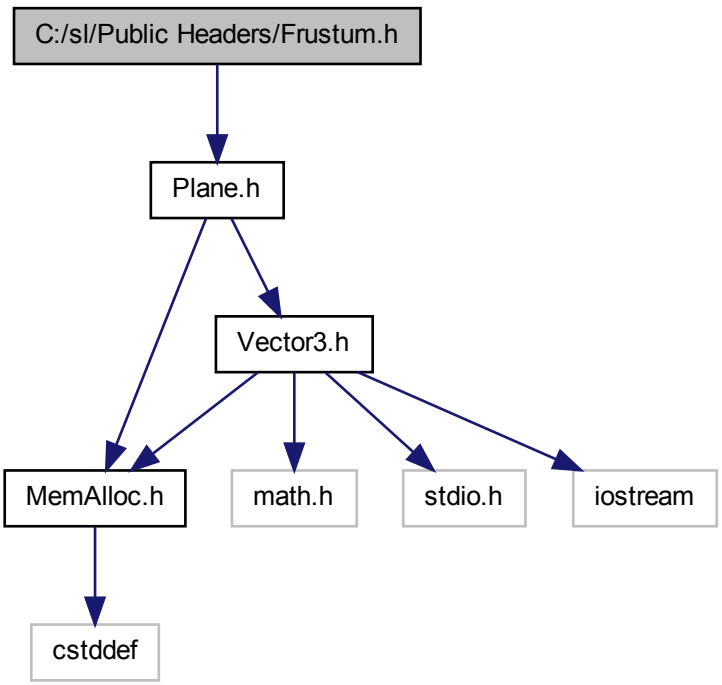
A class that defines an RGBA color and operations on it.

12.7 C:/sl/Public Headers/Frustum.h File Reference

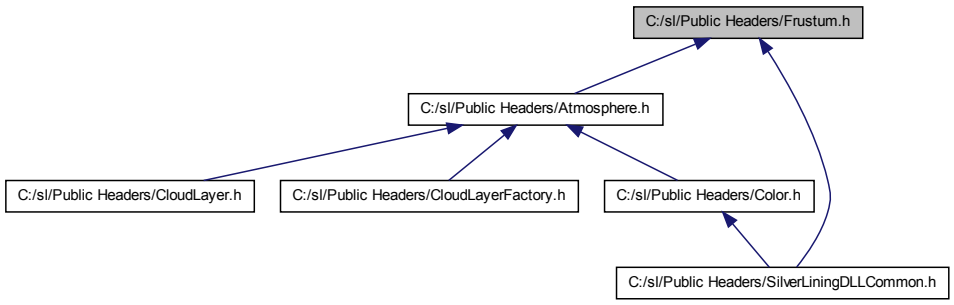
A viewing frustum, modeled as a collection of six planes.

```
#include "Plane.h"
```

Include dependency graph for Frustum.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Frustum

  A simple class that just collects six planes together and calls it a frustum.

12.7.1 Detailed Description

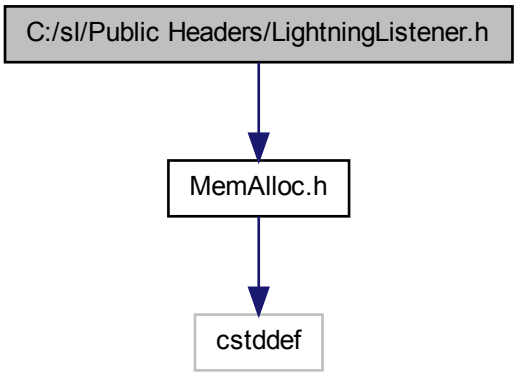
A viewing frustum, modeled as a collection of six planes.

12.8 C:/sl/Public Headers/LightningListener.h File Reference

A virtual base class for receiving lightning strike events in your application.

```
#include "MemAlloc.h"
```

Include dependency graph for LightningListener.h:



Classes

- class SilverLining::LightningListener

  Extend this class in order to receive notifications of lightning strike events from cumulonimbus cloud layers.

12.8.1 Detailed Description

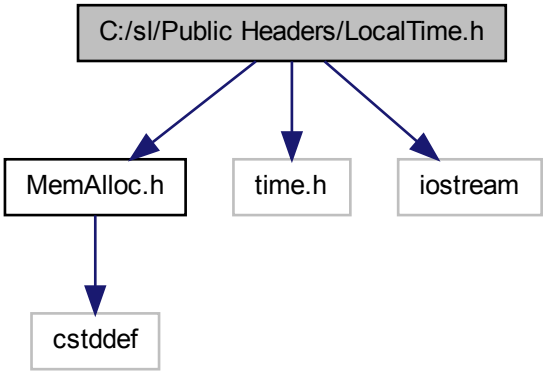
A virtual base class for receiving lightning strike events in your application.

12.9 C:/sl/Public Headers/LocalTime.h File Reference

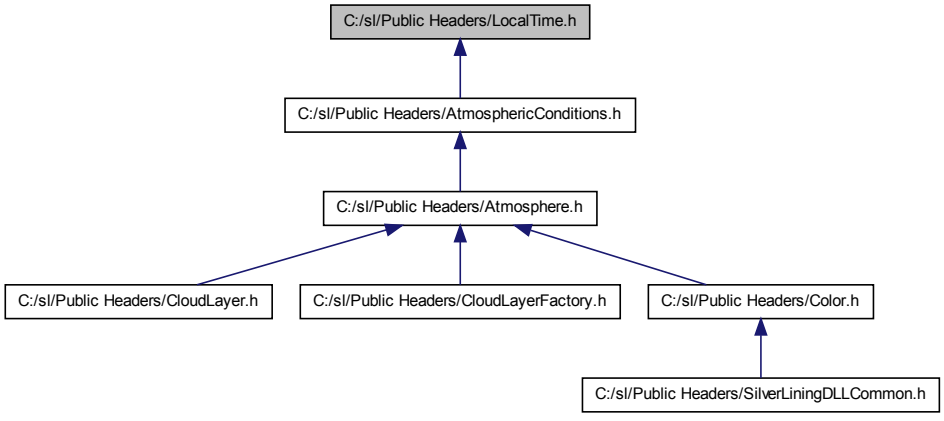
An object to describe the local time, time zone, and daylight savings time observation.

```
#include "MemAlloc.h"
#include <time.h>
#include <iostream>
```

Include dependency graph for LocalTime.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::LocalTime

  A class to represent the simulated local time, time zone, and daylight savings time observation.

Enumerations

- enum TimeZones { ,
  CET = 1, EET = 2, BT = 3,
  GET = 4, PKT = 5, BST = 6,
  THA = 7, CCT = 8, JST = 9,
  GST = 10, SBT = 11, IDLE = 12,
  WAT = -1, AT = -2, BRT = -3,
  AST = -4, EST = -5, CST = -6,
  MST = -7, PST = -8, YST = -9,
  AHST = -10, NT = -11, IDLW = -12 }

  An enumeration of defined time zones worldwide.

12.9.1 Detailed Description

An object to describe the local time, time zone, and daylight savings time observation.

12.9.2 Enumeration Type Documentation

12.9.2.1 enum TimeZones

An enumeration of defined time zones worldwide.

Time zones are expressed as the hour correction (prior to daylight savings adjustments) from GMT. This enum provides names for the civilian time zones, and notes their military equivalents.

Enumerator:

CET ZULU Greenwich Mean Time, UTC, Western European (WET)

EET ALPHA Central European.

BT BETA Eastern European.

GET CHARLIE Baghdad Time.

PKT DELTA Georgia Standard.

BST ECHO Pakistan Standard.

THA FOXTROT Bangladesh Standard.

CCT GOLF Thailand Standard.

JST HOTEL China Coast.

GST INDIA Japan Standard.

SBT KILO Guam Standard.

IDLE LIMA Solomon Islands.

WAT MIKE International Date Line East, NZST (New Zealand Standard)

AT NOVEMBER West Africa.

BRT OSCAR Azores.

AST PAPA Brasilia.

EST QUEBEC Atlantic Standard.

CST ROMEO Eastern Standard.

MST SIERRA Central Standard.

PST TANGO Mountain Standard.

YST UNIFORM Pacific Standard.

AHST VICTOR Yukon Standard.

NT WHISKEY Alaska-Hawaii Standard; Central Alaska (CAT), Hawaii Std (HST)

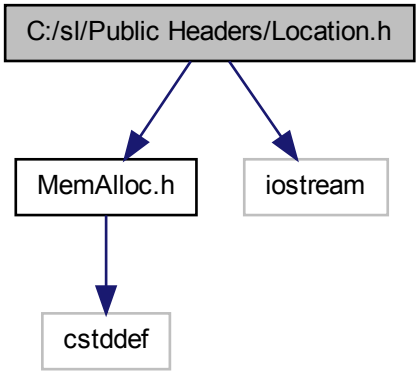
IDLW X-RAY Nome.

12.10 C:/sl/Public Headers/Location.h File Reference

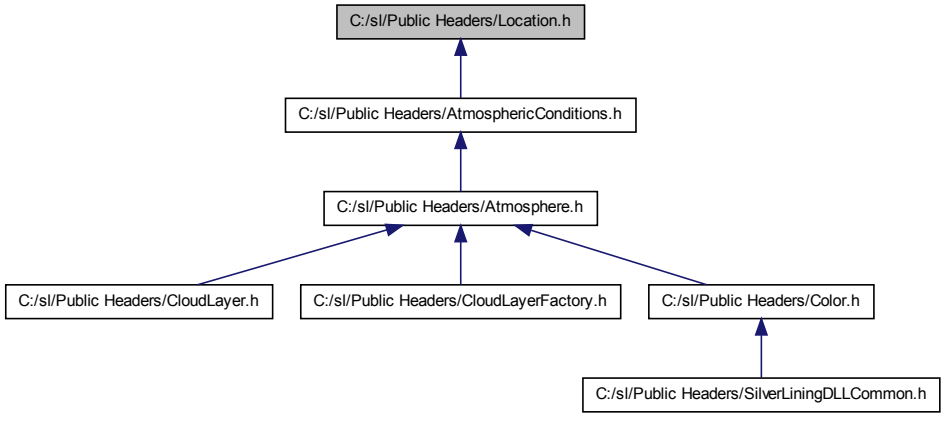
Represents the geographic location being simulated.

```
#include "MemAlloc.h"
#include <iostream>
```

Include dependency graph for Location.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Location

  An object representing a geographic location.

12.10.1 Detailed Description

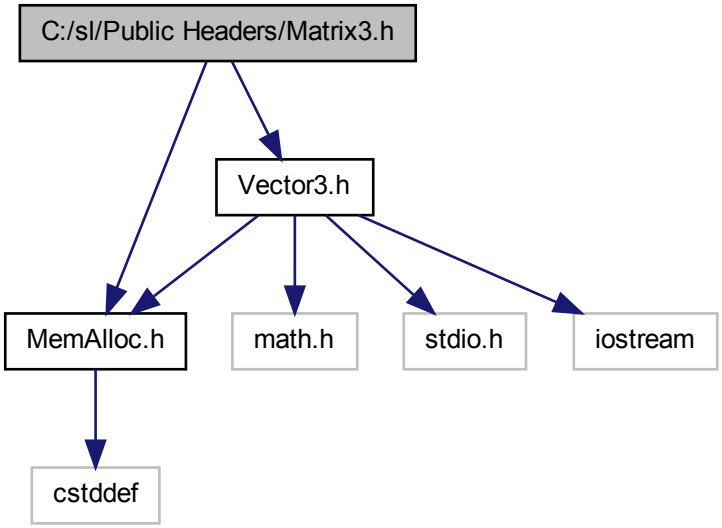
Represents the geographic location being simulated.

12.11 C:/sl/Public Headers/Matrix3.h File Reference

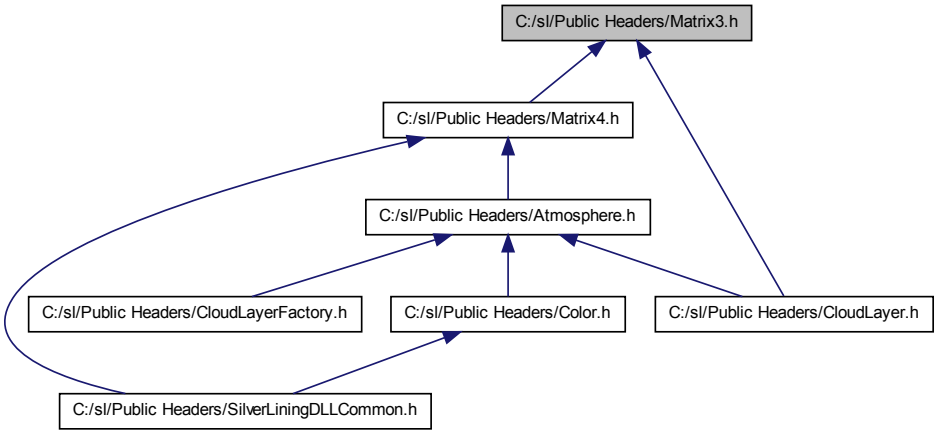
Implements a 3x3 matrix and its operations.

```
#include "MemAlloc.h"
#include "Vector3.h"
```

Include dependency graph for Matrix3.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Matrix3

  A simple 3x3 matrix class and its operations.

12.11.1 Detailed Description

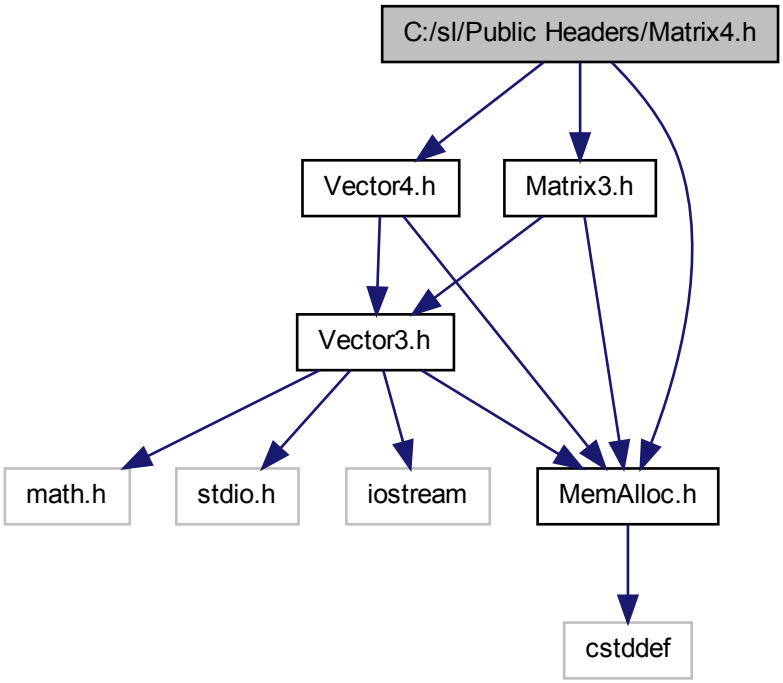
Implements a 3x3 matrix and its operations.

12.12 C:/sl/Public Headers/Matrix4.h File Reference

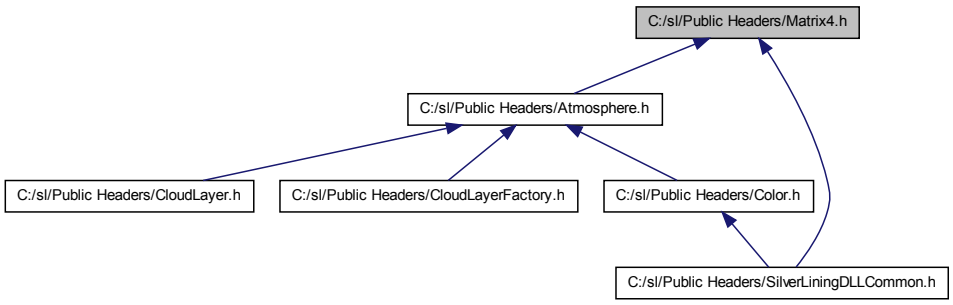
An implementation of a 4x4 matrix and some simple operations on it.

```
#include "MemAlloc.h"
#include "Vector4.h"
#include "Matrix3.h"
```

Include dependency graph for Matrix4.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Matrix4

  An implementation of a 4x4 matrix and some simple operations on it.

12.12.1 Detailed Description

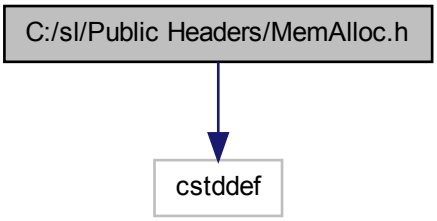
An implementation of a 4x4 matrix and some simple operations on it.

12.13 C:/sl/Public Headers/MemAlloc.h File Reference

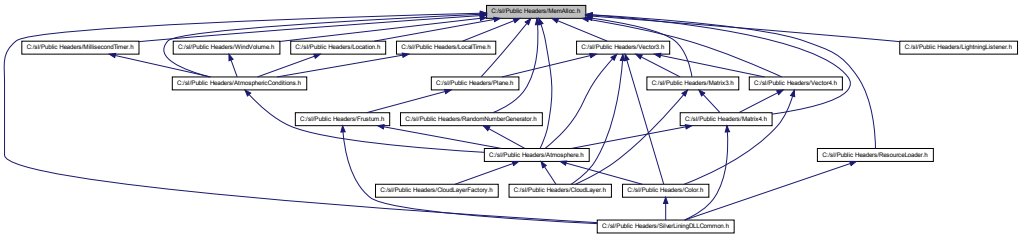
Memory allocation interface for SilverLining.

```
#include <cstddef>
```

Include dependency graph for MemAlloc.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Allocator

  You may extend the Allocator class to hook your own memory management scheme into SilverLining.

- class SilverLining::MemObject

  This base class for all SilverLining objects intercepts the new and delete operators, routing them through SilverLining::Allocator().

12.13.1 Detailed Description

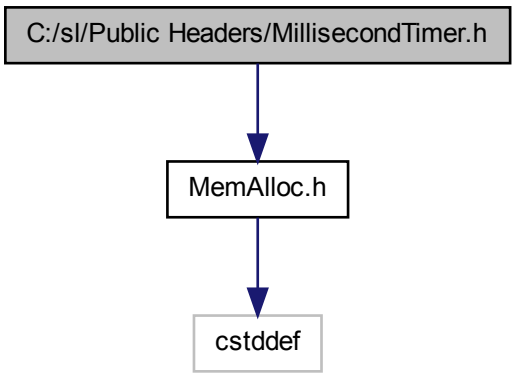
Memory allocation interface for SilverLining.

12.14 C:/sl/Public Headers/MillisecondTimer.h File Reference

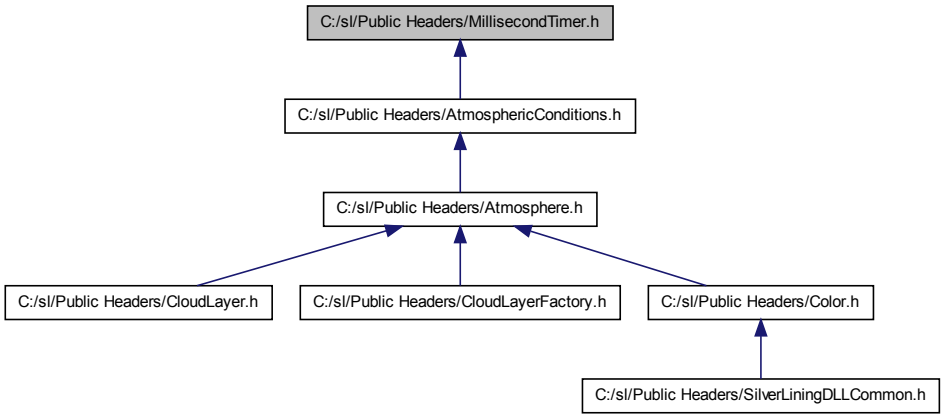
Exposes an interface applications may use to manipulate time.

```
#include "MemAlloc.h"
```

Include dependency graph for MillisecondTimer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::MillisecondTimer

  Exposes an interface to allow applications to control millisecond-level time.

12.14.1 Detailed Description

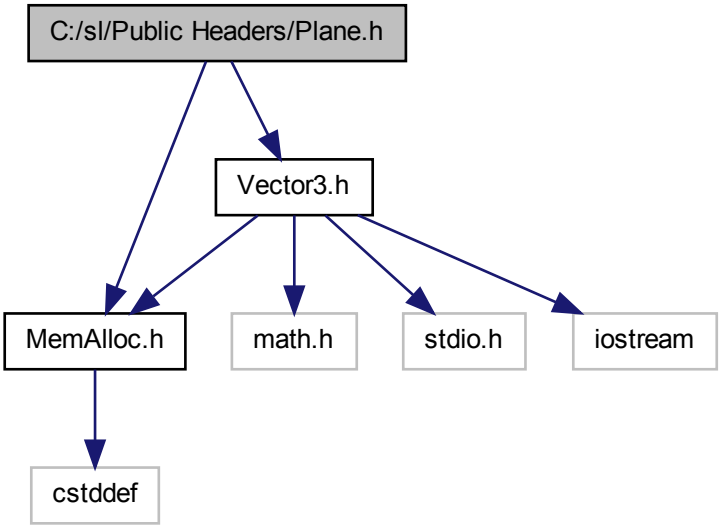
Exposes an interface applications may use to manipulate time.

12.15 C:/sl/Public Headers/Plane.h File Reference

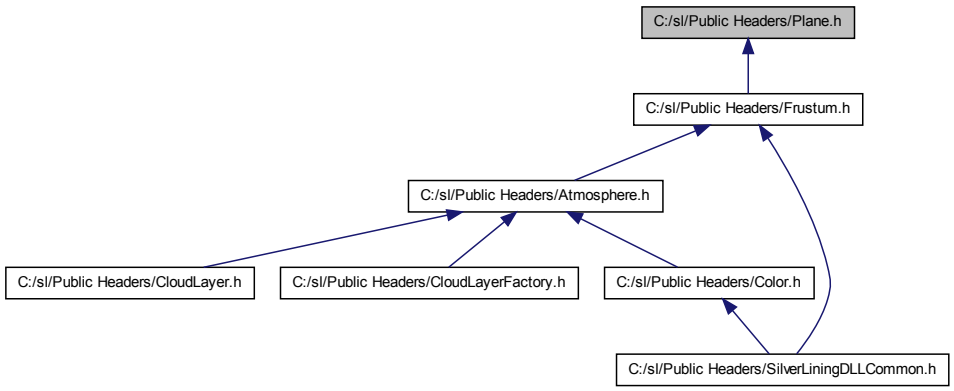
A class that models a geometric plane and its operations.

```
#include "MemAlloc.h"
#include "Vector3.h"
```

Include dependency graph for Plane.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Plane

  A geometric plane, modelled as a normal vector and a distance to the plane.

12.15.1 Detailed Description

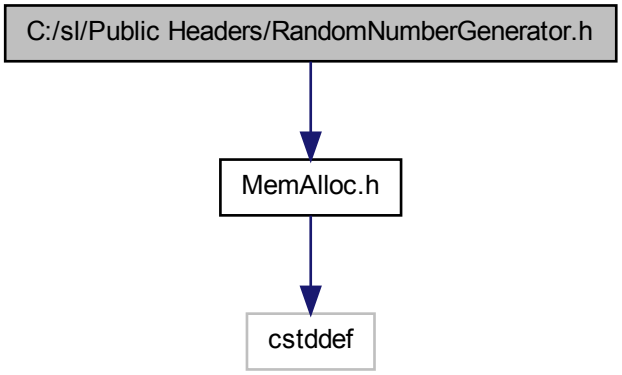
A class that models a geometric plane and its operations.

12.16 C:/sl/Public Headers/RandomNumberGenerator.h File Reference

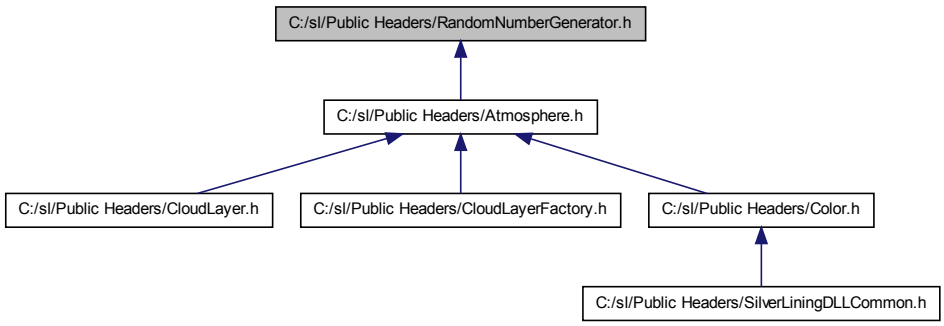
A virtual base class for overriding SilverLining's random number generator in your application.

```
#include "MemAlloc.h"
```

Include dependency graph for RandomNumberGenerator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::RandomNumberGenerator

Extend this class in order to use your own random number generator with SilverLining.

12.16.1 Detailed Description

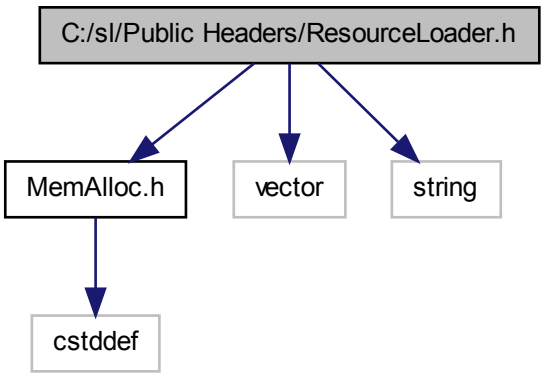
A virtual base class for overriding SilverLining's random number generator in your application.

12.17 C:/sl/Public Headers/ResourceLoader.h File Reference

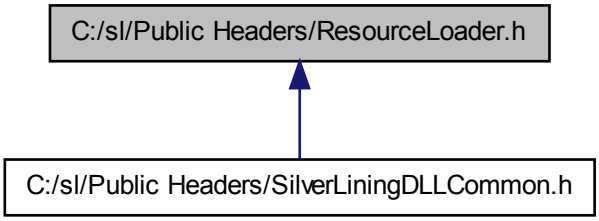
A class for loading SilverLining's resources from mass storage, which you may extend.

```
#include "MemAlloc.h"
#include <vector>
#include <string>
```

Include dependency graph for ResourceLoader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::ResourceLoader

  This class is used whenever SilverLining needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish.

12.17.1 Detailed Description

A class for loading SilverLining's resources from mass storage, which you may extend.

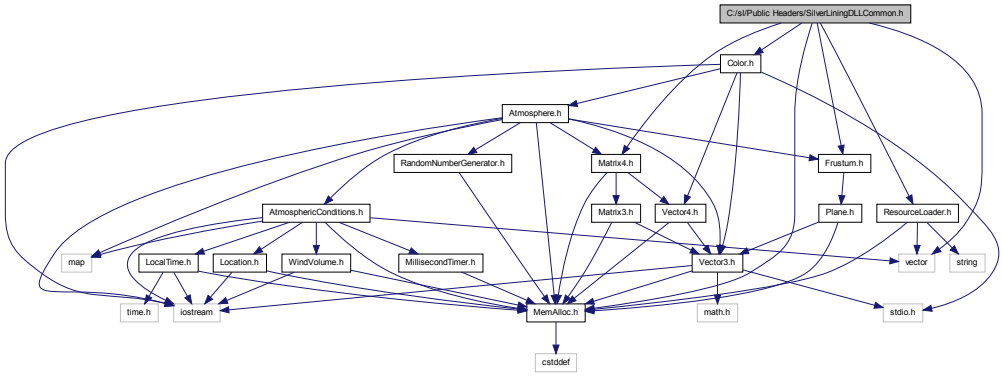
12.18 C:/sl/Public Headers/SilverLiningDLLCommon.h File Reference

Shared header for the specific renderer DLL's, and for use by the Renderer class of the engine.

```
#include "MemAlloc.h"
#include "SilverLiningTypes.h"
#include "Color.h"
#include "Frustum.h"
#include "Matrix4.h"
```

```
#include "Vertex.h"
#include "ResourceLoader.h"
#include <vector>
```

Include dependency graph for SilverLiningDLLCommon.h:



Functions

- SILVERLININGDLL_API void SetUserDefinedVertString (const char *userDefinedString)

  Sets a user defined string to be prepended to all vertex shaders.

- SILVERLININGDLL_API void SetUserDefinedFragString (const char *userDefinedString)

  Sets a user defined string to be prepended to all fragment shaders.

- SILVERLININGDLL_API const char * GetUserDefinedVertString ()

  Retrieves the user defined vertex string previously set with SetUserDefinedVertString.

- SILVERLININGDLL_API const char * GetUserDefinedFragString ()

  Retrieves the user defined fragment string previously set with SetUserDefinedFragString.

12.18.1 Detailed Description

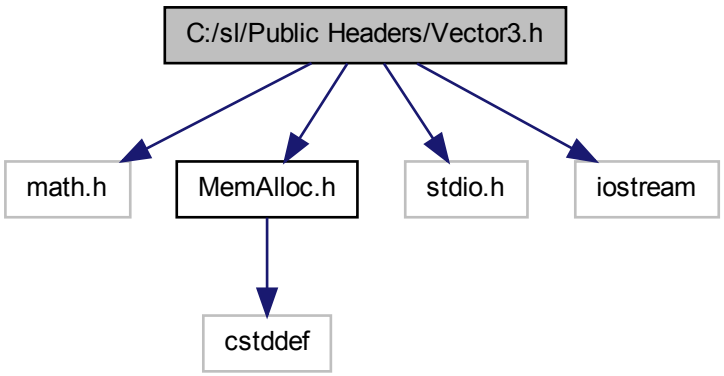
Shared header for the specific renderer DLL's, and for use by the Renderer class of the engine.

12.19 C:/sl/Public Headers/Vector3.h File Reference

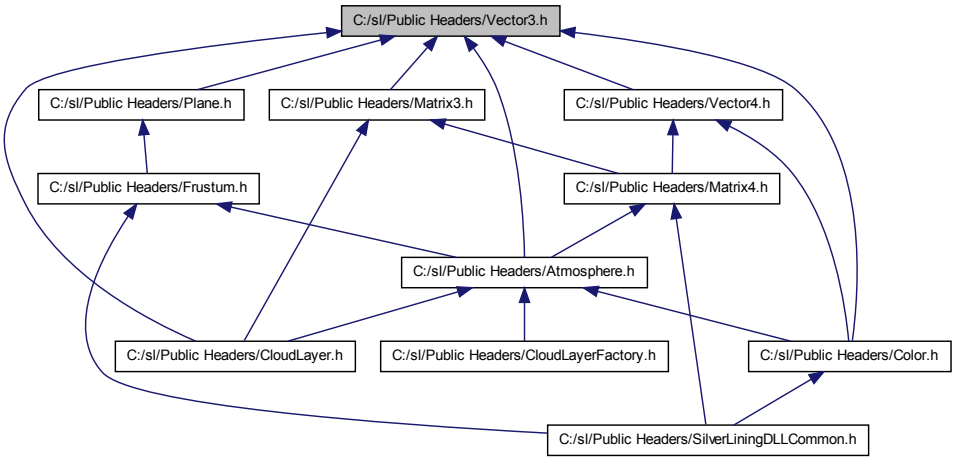
A 3D Vector class and its operations.

```
#include <math.h>
#include "MemAlloc.h"
#include <stdio.h>
#include <iostream>
```

Include dependency graph for Vector3.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Vector3

  A 3D double-precision Vector class and its operations.

- class SilverLining::Vector3f

  A 3D single-precision vector class, and its operations.

12.19.1 Detailed Description

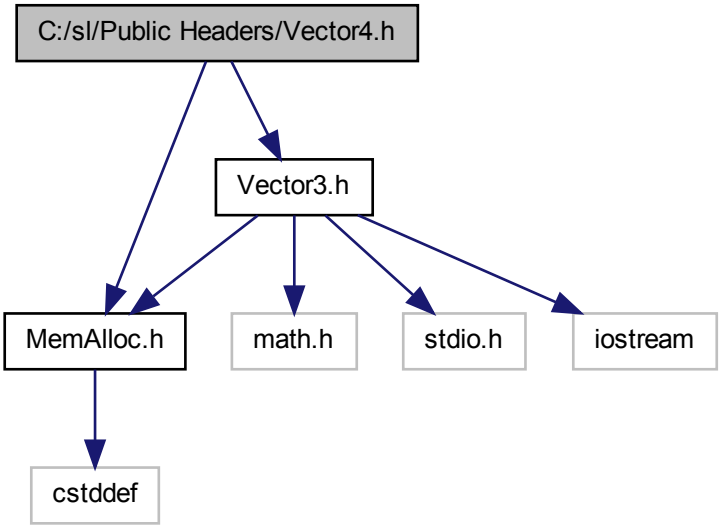
A 3D Vector class and its operations.

12.20 C:/sl/Public Headers/Vector4.h File Reference

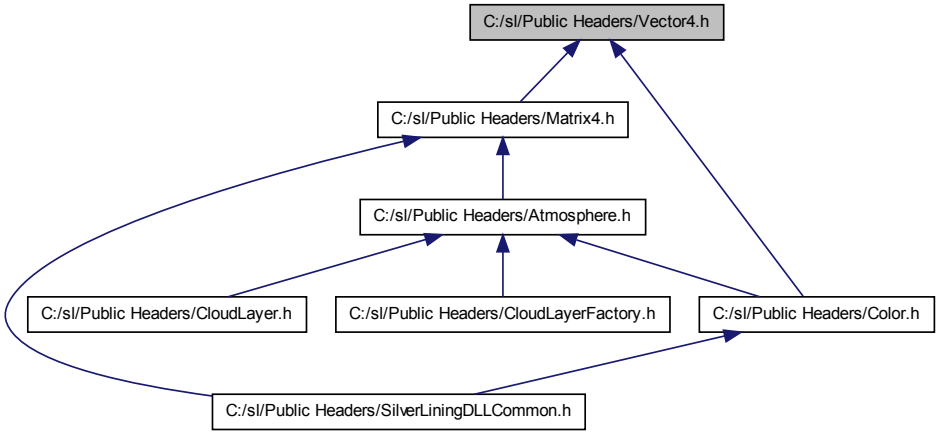
A simple 4D vector class.

```
#include "MemAlloc.h"
#include "Vector3.h"
```

Include dependency graph for Vector4.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::Vector4

  A simple double-precision 4D vector class with no operations defined.

12.20.1 Detailed Description

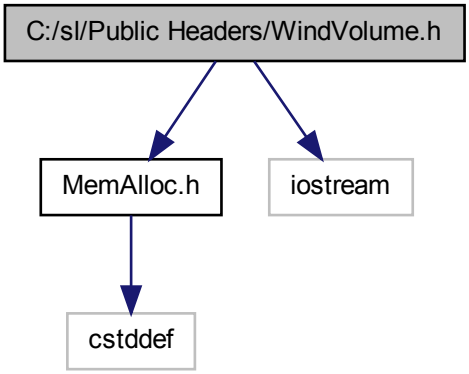
A simple 4D vector class.

12.21 C:/sl/Public Headers/WindVolume.h File Reference

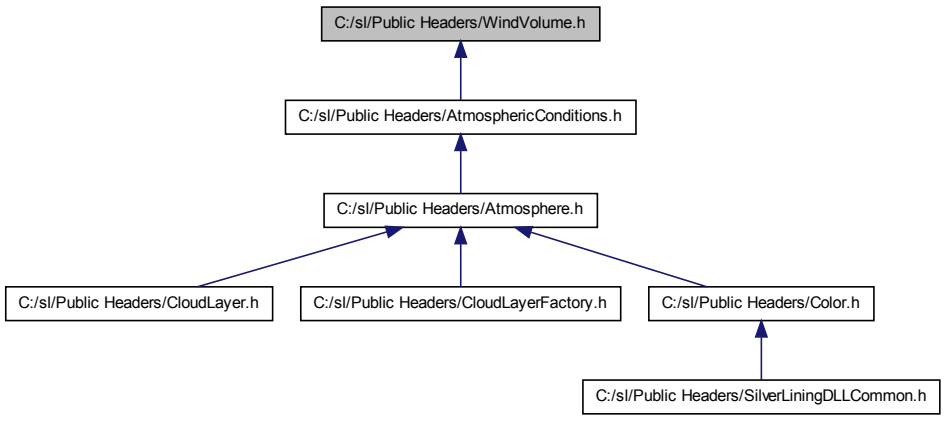
Defines an area of a given wind speed and direction.

```
#include "MemAlloc.h"
#include <iostream>
```

Include dependency graph for WindVolume.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SilverLining::WindVolume

  Defines an area of a given wind velocity and direction bounded by two altitudes.

12.21.1 Detailed Description

Defines an area of a given wind speed and direction.

Index

~Atmosphere
SilverLining::Atmosphere, 68
~AtmosphericConditions
SilverLining::AtmosphericConditions, 98
~CloudLayer
SilverLining::CloudLayer, 114
~Frustum
SilverLining::Frustum, 138
~LightningListener
SilverLining::LightningListener, 140
~LocalTime
SilverLining::LocalTime, 144
~Matrix3
SilverLining::Matrix3, 156
~Matrix4
SilverLining::Matrix4, 160
~MillisecondTimer
SilverLining::MillisecondTimer, 165
~RandomNumberGenerator
SilverLining::RandomNumberGenerator, 169
~WindVolume
SilverLining::WindVolume, 187

AddCloudLayer
SilverLining::AtmosphericConditions, 98
AddLightningListener
SilverLining::CloudLayer, 114
AddSeconds
SilverLining::LocalTime, 144
AHST
LocalTime.h, 205
ApplyWind
SilverLining::AtmosphericConditions, 98
AST
LocalTime.h, 205
AT
LocalTime.h, 205
Atmosphere
SilverLining::Atmosphere, 67
AtmosphericConditions
SilverLining::AtmosphericConditions, 98

BRT
LocalTime.h, 205
BST
LocalTime.h, 205
BT
LocalTime.h, 204

C:/sl/Public Headers/Atmosphere.h, 191
C:/sl/Public Headers/AtmosphericConditions.h, 193
C:/sl/Public Headers/CloudLayer.h, 194
C:/sl/Public Headers/CloudLayerFactory.h, 195
C:/sl/Public Headers/CloudTypes.h, 196
C:/sl/Public Headers/Color.h, 198
C:/sl/Public Headers/Frustum.h, 199
C:/sl/Public Headers/LightningListener.h, 201
C:/sl/Public Headers/LocalTime.h, 202
C:/sl/Public Headers/Location.h, 205
C:/sl/Public Headers/Matrix3.h, 207
C:/sl/Public Headers/Matrix4.h, 208
C:/sl/Public Headers/MemAlloc.h, 210
C:/sl/Public Headers/MillisecondTimer.h, 212
C:/sl/Public Headers/Plane.h, 213

C:/sl/Public Headers/RandomNumberGenerator.h, 215
C:/sl/Public Headers/ResourceLoader.h, 217
C:/sl/Public Headers/SilverLiningDLLCommon.h, 218
C:/sl/Public Headers/Vector3.h, 219
C:/sl/Public Headers/Vector4.h, 221
C:/sl/Public Headers/WindVolume.h, 223
CCT
    LocalTime.h, 205
CET
    LocalTime.h, 204
CIRROCUMULUS
    CloudTypes.h, 197
CIRRUS_FIBRATUS
    CloudTypes.h, 197
ClampToUnitOrLess
    SilverLining::Color, 135
ClearCloudLightingOverride
    SilverLining::Atmosphere, 68
ClearFog
    SilverLining::AtmosphericConditions, 99
ClearLightningListeners
    SilverLining::CloudLayer, 114
ClearWindVolumes
    SilverLining::AtmosphericConditions, 99
CloudLayer
    SilverLining::CloudLayer, 114
CloudTypes
    CloudTypes.h, 197
CloudTypes.h
    CIRROCUMULUS, 197
    CIRRUS_FIBRATUS, 197
    CloudTypes, 197
    CUMULONIMBUS_CAPPILATUS, 198
    CUMULUS_CONGESTUS, 197
    CUMULUS_MEDIOCRIS, 197
    NUM_CLOUD_TYPES, 198
    STRATOCUMULUS, 198
    STRATUS, 197
Color
    SilverLining::Color, 134, 135
Create
    SilverLining::CloudLayerFactory, 131
Cross
    SilverLining::Vector3, 175
CST
    LocalTime.h, 205
CullObjects
    SilverLining::Atmosphere, 68
CUMULONIMBUS_CAPPILATUS
    CloudTypes.h, 198
CUMULUS_CONGESTUS
    CloudTypes.h, 197
CUMULUS_MEDIOCRIS
    CloudTypes.h, 197

DisableFarCulling
    SilverLining::Atmosphere, 68
Dot
    SilverLining::Vector3, 175
    SilverLining::Vector3f, 179
    SilverLining::Vector4, 182
DrawObject
    SilverLining::Atmosphere, 69
DrawObjects
    SilverLining::Atmosphere, 69
DrawSky
    SilverLining::Atmosphere, 70

EET
    LocalTime.h, 204
EnableHDR
    SilverLining::Atmosphere, 72
EnableLensFlare
    SilverLining::Atmosphere, 72
EnableTimePassage
    SilverLining::AtmosphericConditions, 99
EST
    LocalTime.h, 205
ExportToVRML
    SilverLining::CloudLayer, 115

ForceLightingRecompute
    SilverLining::Atmosphere, 72
ForceLightning
    SilverLining::CloudLayer, 115
FreeResource

SilverLining::ResourceLoader, 171
FromRx
SilverLining::Matrix3, 157
FromRy
SilverLining::Matrix3, 157
FromRz
SilverLining::Matrix3, 157
FromXYZ
SilverLining::Matrix3, 157
Frustum
SilverLining::Frustum, 138

GET
LocalTime.h, 204
GetAlpha
SilverLining::CloudLayer, 115
GetAltitude
SilverLining::Location, 152
GetAmbientColor
SilverLining::Atmosphere, 73
GetBaseAltitude
SilverLining::CloudLayer, 115
GetBaseLength
SilverLining::CloudLayer, 115
GetBaseWidth
SilverLining::CloudLayer, 116
GetCameraMatrix
SilverLining::Atmosphere, 73
GetCloudBounds
SilverLining::Atmosphere, 73
GetCloudLayer
SilverLining::AtmosphericConditions, 100
GetCloudWrapping
SilverLining::CloudLayer, 116
GetConditions
SilverLining::Atmosphere, 74
GetConfigOptionBoolean
SilverLining::Atmosphere, 74
GetConfigOptionDouble
SilverLining::Atmosphere, 74
GetConfigOptionString
SilverLining::Atmosphere, 74
GetDay
SilverLining::LocalTime, 144
GetDensity
SilverLining::CloudLayer, 116
GetDensityMultiplier
SilverLining::CloudLayer, 116
GetDirection
SilverLining::WindVolume, 187
GetDistance
SilverLining::Plane, 168
GetEnabled
SilverLining::CloudLayer, 116
GetEnvironmentMap
SilverLining::Atmosphere, 75
GetEpoch1990Days
SilverLining::LocalTime, 144
GetEpoch2000Centuries
SilverLining::LocalTime, 145
GetFadeTowardEdges
SilverLining::CloudLayer, 117
GetFarCullingDisabled
SilverLining::Atmosphere, 75
GetFilesInDirectory
SilverLining::ResourceLoader, 171
GetFog
SilverLining::AtmosphericConditions, 100
GetFogEnabled
SilverLining::Atmosphere, 75
GetFogSettings
SilverLining::Atmosphere, 76
GetFramerate
SilverLining::Atmosphere, 76
GetGamma
SilverLining::Atmosphere, 76
GetHaze
SilverLining::Atmosphere, 76
GetHDREnabled
SilverLining::Atmosphere, 77
GetHorizonColor
SilverLining::Atmosphere, 77
GetHour
SilverLining::LocalTime, 145
GetInfraRedMode
SilverLining::Atmosphere, 78
GetIsInfinite
SilverLining::CloudLayer, 117
GetJulianDate
SilverLining::LocalTime, 145

GetLatitude
SilverLining::Location, 152
GetLayerPosition
SilverLining::CloudLayer, 117
GetLensFlareEnabled
SilverLining::Atmosphere, 78
GetLightPollution
SilverLining::AtmosphericConditions, 100
GetLocation
SilverLining::AtmosphericConditions, 100
GetLongitude
SilverLining::Location, 152
GetMaxAltitude
SilverLining::WindVolume, 187
GetMaxHeight
SilverLining::CloudLayer, 117
GetMilliseconds
SilverLining::MillisecondTimer, 165
GetMillisecondTimer
SilverLining::AtmosphericConditions, 101
GetMinAltitude
SilverLining::WindVolume, 187
GetMinutes
SilverLining::LocalTime, 146
GetMonth
SilverLining::LocalTime, 146
GetMoonAlpha
SilverLining::Atmosphere, 78
GetMoonColor
SilverLining::Atmosphere, 78
GetMoonPosition
SilverLining::Atmosphere, 79
GetMoonPositionEquatorial
SilverLining::Atmosphere, 79
GetMoonPositionGeographic
SilverLining::Atmosphere, 79
GetNormal
SilverLining::Plane, 168
GetObjectDistance
SilverLining::Atmosphere, 80
GetObjectPosition
SilverLining::Atmosphere, 81
GetObservingDaylightSavingsTime
SilverLining::LocalTime, 146
GetPlane
SilverLining::Frustum, 138
GetPrecipitationWind
SilverLining::AtmosphericConditions, 101
GetProjectionMatrix
SilverLining::Atmosphere, 81
GetResourceLoader
SilverLining::Atmosphere, 81
GetRightVector
SilverLining::Atmosphere, 81
GetRow
SilverLining::Matrix4, 161
GetSeconds
SilverLining::LocalTime, 146
GetShadowMap
SilverLining::Atmosphere, 82
GetSkyCoverage
SilverLining::Atmosphere, 83
GetSunAlpha
SilverLining::Atmosphere, 83
GetSunColor
SilverLining::Atmosphere, 83
GetSunOcclusion
SilverLining::Atmosphere, 84
GetSunOrMoonColor
SilverLining::Atmosphere, 84
GetSunOrMoonPosition
SilverLining::Atmosphere, 85
GetSunOrMoonPositionEquatorial
SilverLining::Atmosphere, 85
GetSunOrMoonPositionGeographic
SilverLining::Atmosphere, 85
GetSunPosition
SilverLining::Atmosphere, 85
GetSunPositionEquatorial
SilverLining::Atmosphere, 85
GetSunPositionGeographic
SilverLining::Atmosphere, 86
GetThickness
SilverLining::CloudLayer, 117
GetTime
SilverLining::AtmosphericConditions, 101
GetTimeZone

SilverLining::LocalTime, 146
GetTurbidity
SilverLining::AtmosphericConditions, 101
GetType
SilverLining::CloudLayer, 118
GetUnitScale
SilverLining::Atmosphere, 86
GetUpVector
SilverLining::Atmosphere, 86
GetVisibility
SilverLining::AtmosphericConditions, 102
GetWind
SilverLining::AtmosphericConditions, 102
SilverLining::CloudLayer, 118
GetWindSpeed
SilverLining::WindVolume, 187
GetYear
SilverLining::LocalTime, 147
GetZenithColor
SilverLining::Atmosphere, 86
GST
LocalTime.h, 205

HasPrecipitationAtPosition
SilverLining::CloudLayer, 118

IDLE
LocalTime.h, 205
IDLW
LocalTime.h, 205
Initialize
SilverLining::Atmosphere, 86
Inside
SilverLining::WindVolume, 188
IsInitialized
SilverLining::Atmosphere, 88
IsInsideCloud
SilverLining::CloudLayer, 118
IsRenderable
SilverLining::CloudLayer, 119

JST
LocalTime.h, 205

Length
SilverLining::Vector3, 175
SilverLining::Vector3f, 179
LightningStartedEvent
SilverLining::LightningListener, 140
LoadResource
SilverLining::ResourceLoader, 171
LocalTime
SilverLining::LocalTime, 144
LocalTime.h
AHST, 205
AST, 205
AT, 205
BRT, 205
BST, 205
BT, 204
CCT, 205
CET, 204
CST, 205
EET, 204
EST, 205
GET, 204
GST, 205
IDLE, 205
IDLW, 205
JST, 205
MST, 205
NT, 205
PKT, 204
PST, 205
SBT, 205
THA, 205
TimeZones, 204
WAT, 205
YST, 205
Location
SilverLining::Location, 152

Matrix3
SilverLining::Matrix3, 156
Matrix4
SilverLining::Matrix4, 160
MillisecondTimer
SilverLining::MillisecondTimer, 165
MST
LocalTime.h, 205

Normalize
SilverLining::Plane, 168
SilverLining::Vector3, 175
SilverLining::Vector3f, 179
NT
LocalTime.h, 205
NUM_CLOUD_TYPES
CloudTypes.h, 198

operator∗
SilverLining::Color, 135
SilverLining::Matrix3, 157, 158
SilverLining::Matrix4, 161
SilverLining::Vector3, 175
SilverLining::Vector3f, 179
SilverLining::Vector4, 182
operator+
SilverLining::Color, 135
SilverLining::Vector3, 175
SilverLining::Vector3f, 179
operator-
SilverLining::Vector3, 176
SilverLining::Vector3f, 179
operator==
SilverLining::Vector3, 176
OverrideCloudLighting
SilverLining::Atmosphere, 88

PKT
LocalTime.h, 204
Plane
SilverLining::Plane, 167
Planes
SilverLining::Frustum, 138
PST
LocalTime.h, 205

ReloadConfigFile
SilverLining::Atmosphere, 88
RemoveAllCloudLayers
SilverLining::AtmosphericConditions, 102
RemoveCloudLayer
SilverLining::AtmosphericConditions, 102
RemoveWindVolume
SilverLining::AtmosphericConditions, 103
Restore
SilverLining::CloudLayer, 119

Save
SilverLining::CloudLayer, 120
SBT
LocalTime.h, 205
ScaleToUnitOrLess
SilverLining::Color, 135
SeedClouds
SilverLining::CloudLayer, 120
Serialize
SilverLining::Atmosphere, 89
SilverLining::AtmosphericConditions, 103
SilverLining::CloudLayer, 121
SilverLining::CloudLayerFactory, 131
SilverLining::Color, 135
SilverLining::LocalTime, 147
SilverLining::Location, 153
SilverLining::Vector3, 176
SilverLining::WindVolume, 188
SetAllocator
SilverLining::Allocator, 58
SetAlpha
SilverLining::CloudLayer, 121
SetAltitude
SilverLining::Location, 153
SetBaseAltitude
SilverLining::CloudLayer, 121
SetBaseLength
SilverLining::CloudLayer, 121
SetBaseWidth
SilverLining::CloudLayer, 122
SetCameraMatrix
SilverLining::Atmosphere, 89
SetCloudAnimationEffects
SilverLining::CloudLayer, 122
SetCloudWrapping
SilverLining::CloudLayer, 123
SetColor
SilverLining::Vertex, 184
SetConditions
SilverLining::Atmosphere, 89

SetConfigOption
SilverLining::Atmosphere, 89
SetDay
SilverLining::LocalTime, 147
SetDensity
SilverLining::CloudLayer, 123
SetDensityMultiplier
SilverLining::CloudLayer, 124
SetDirection
SilverLining::WindVolume, 188
SetDistance
SilverLining::Plane, 168
SetEnabled
SilverLining::CloudLayer, 124
SetFadeTowardEdges
SilverLining::CloudLayer, 125
SetFog
SilverLining::AtmosphericConditions, 103
SetFromEpochSeconds
SilverLining::LocalTime, 147
SetFromSystemTime
SilverLining::LocalTime, 148
SetGamma
SilverLining::Atmosphere, 89
SetHaze
SilverLining::Atmosphere, 90
SetHour
SilverLining::LocalTime, 148
SetInfraRedMode
SilverLining::Atmosphere, 90
SetIsInfinite
SilverLining::CloudLayer, 125
SetLatitude
SilverLining::Location, 153
SetLayerPosition
SilverLining::CloudLayer, 126
SetLightPollution
SilverLining::AtmosphericConditions, 103
SetLocation
SilverLining::AtmosphericConditions, 103
SetLongitude
SilverLining::Location, 153
SetMaxAltitude
SilverLining::WindVolume, 188
SetMillisecondTimer
SilverLining::AtmosphericConditions, 104
SetMinAltitude
SilverLining::WindVolume, 189
SetMinutes
SilverLining::LocalTime, 148
SetMonth
SilverLining::LocalTime, 148
SetMoonAlpha
SilverLining::Atmosphere, 90
SetNormal
SilverLining::Plane, 168
SetObservingDaylightSavingsTime
SilverLining::LocalTime, 149
SetPlane
SilverLining::Frustum, 139
SetPrecipitation
SilverLining::AtmosphericConditions, 104
SilverLining::CloudLayer, 126
SetPrecipitationWind
SilverLining::AtmosphericConditions, 105
SetPresetConditions
SilverLining::AtmosphericConditions, 105
SetProjectionMatrix
SilverLining::Atmosphere, 91
SetRandomNumberGenerator
SilverLining::Atmosphere, 91
SetResourceDirPath
SilverLining::ResourceLoader, 172
SetResourceLoader
SilverLining::Atmosphere, 91
SetRightVector
SilverLining::Atmosphere, 91
SetSeconds
SilverLining::LocalTime, 149
SetSunAlpha
SilverLining::Atmosphere, 92
SetThickness
SilverLining::CloudLayer, 127
SetTime

SilverLining::AtmosphericConditions, 106
SetTimeZone
SilverLining::LocalTime, 149
SetTurbidity
SilverLining::AtmosphericConditions, 106
SetType
SilverLining::CloudLayer, 128
SetUpVector
SilverLining::Atmosphere, 92
SetUV
SilverLining::Vertex, 184
SetVisibility
SilverLining::AtmosphericConditions, 107
SetWind
SilverLining::AtmosphericConditions, 107
SilverLining::CloudLayer, 128
SetWindSpeed
SilverLining::WindVolume, 189
SetWorldUnits
SilverLining::Atmosphere, 92
SetYear
SilverLining::LocalTime, 150
SilverLining::Allocator, 57
SetAllocator, 58
SilverLining::Atmosphere, 58
~Atmosphere, 68
Atmosphere, 67
ClearCloudLightingOverride, 68
CullObjects, 68
DisableFarCulling, 68
DrawObject, 69
DrawObjects, 69
DrawSky, 70
EnableHDR, 72
EnableLensFlare, 72
ForceLightingRecompute, 72
GetAmbientColor, 73
GetCameraMatrix, 73
GetCloudBounds, 73
GetConditions, 74
GetConfigOptionBoolean, 74
GetConfigOptionDouble, 74
GetConfigOptionString, 74
GetEnvironmentMap, 75
GetFarCullingDisabled, 75
GetFogEnabled, 75
GetFogSettings, 76
GetFramerate, 76
GetGamma, 76
GetHaze, 76
GetHDREnabled, 77
GetHorizonColor, 77
GetInfraRedMode, 78
GetLensFlareEnabled, 78
GetMoonAlpha, 78
GetMoonColor, 78
GetMoonPosition, 79
GetMoonPositionEquatorial, 79
GetMoonPositionGeographic, 79
GetObjectDistance, 80
GetObjectPosition, 81
GetProjectionMatrix, 81
GetResourceLoader, 81
GetRightVector, 81
GetShadowMap, 82
GetSkyCoverage, 83
GetSunAlpha, 83
GetSunColor, 83
GetSunOcclusion, 84
GetSunOrMoonColor, 84
GetSunOrMoonPosition, 85
GetSunOrMoonPositionEquatorial, 85
GetSunOrMoonPositionGeographic, 85
GetSunPosition, 85
GetSunPositionEquatorial, 85
GetSunPositionGeographic, 86
GetUnitScale, 86
GetUpVector, 86
GetZenithColor, 86
Initialize, 86
IsInitialized, 88
OverrideCloudLighting, 88
ReloadConfigFile, 88
Serialize, 89
SetCameraMatrix, 89
SetConditions, 89
SetConfigOption, 89
SetGamma, 89

SetHaze, 90
SetInfraRedMode, 90
SetMoonAlpha, 90
SetProjectionMatrix, 91
SetRandomNumberGenerator, 91
SetResourceLoader, 91
SetRightVector, 91
SetSunAlpha, 92
SetUpVector, 92
SetWorldUnits, 92
SL_VECTOR, 92
Unserialize, 93
UpdateEphemeris, 93
UpdateSkyAndClouds, 93
SilverLining::AtmosphericConditions, 94
~AtmosphericConditions, 98
AddCloudLayer, 98
ApplyWind, 98
AtmosphericConditions, 98
ClearFog, 99
ClearWindVolumes, 99
EnableTimePassage, 99
GetCloudLayer, 100
GetFog, 100
GetLightPollution, 100
GetLocation, 100
GetMillisecondTimer, 101
GetPrecipitationWind, 101
GetTime, 101
GetTurbidity, 101
GetVisibility, 102
GetWind, 102
RemoveAllCloudLayers, 102
RemoveCloudLayer, 102
RemoveWindVolume, 103
Serialize, 103
SetFog, 103
SetLightPollution, 103
SetLocation, 103
SetMillisecondTimer, 104
SetPrecipitation, 104
SetPrecipitationWind, 105
SetPresetConditions, 105
SetTime, 106
SetTurbidity, 106
SetVisibility, 107
SetWind, 107
SL_MAP, 107
SilverLining::CloudLayer, 108
~CloudLayer, 114
AddLightningListener, 114
ClearLightningListeners, 114
CloudLayer, 114
ExportToVRML, 115
ForceLightning, 115
GetAlpha, 115
GetBaseAltitude, 115
GetBaseLength, 115
GetBaseWidth, 116
GetCloudWrapping, 116
GetDensity, 116
GetDensityMultiplier, 116
GetEnabled, 116
GetFadeTowardEdges, 117
GetIsInfinite, 117
GetLayerPosition, 117
GetMaxHeight, 117
GetThickness, 117
GetType, 118
GetWind, 118
HasPrecipitationAtPosition, 118
IsInsideCloud, 118
IsRenderable, 119
Restore, 119
Save, 120
SeedClouds, 120
Serialize, 121
SetAlpha, 121
SetBaseAltitude, 121
SetBaseLength, 121
SetBaseWidth, 122
SetCloudAnimationEffects, 122
SetCloudWrapping, 123
SetDensity, 123
SetDensityMultiplier, 124
SetEnabled, 124
SetFadeTowardEdges, 125
SetIsInfinite, 125
SetLayerPosition, 126
SetPrecipitation, 126
SetThickness, 127
SetType, 128

SetWind, 128
SL_MAP, 128
SilverLining::CloudLayerFactory, 129
Create, 131
Serialize, 131
SilverLining::Color, 131
ClampToUnitOrLess, 135
Color, 134, 135
operator∗, 135
operator+, 135
ScaleToUnitOrLess, 135
Serialize, 135
ToGrayscale, 135
Unserialize, 136
SilverLining::Frustum, 136
∼Frustum, 138
Frustum, 138
GetPlane, 138
Planes, 138
SetPlane, 139
SilverLining::LightningListener, 139
∼LightningListener, 140
LightningStartedEvent, 140
SilverLining::LocalTime, 140
∼LocalTime, 144
AddSeconds, 144
GetDay, 144
GetEpoch1990Days, 144
GetEpoch2000Centuries, 145
GetHour, 145
GetJulianDate, 145
GetMinutes, 146
GetMonth, 146
GetObservingDaylightSavingsTime, 146
GetSeconds, 146
GetTimeZone, 146
GetYear, 147
LocalTime, 144
Serialize, 147
SetDay, 147
SetFromEpochSeconds, 147
SetFromSystemTime, 148
SetHour, 148
SetMinutes, 148
SetMonth, 148
SetObservingDaylightSavingsTime, 149
SetSeconds, 149
SetTimeZone, 149
SetYear, 150
SilverLining::Location, 150
GetAltitude, 152
GetLatitude, 152
GetLongitude, 152
Location, 152
Serialize, 153
SetAltitude, 153
SetLatitude, 153
SetLongitude, 153
SilverLining::Matrix3, 154
∼Matrix3, 156
FromRx, 157
FromRy, 157
FromRz, 157
FromXYZ, 157
Matrix3, 156
operator∗, 157, 158
Transpose, 157
SilverLining::Matrix4, 158
∼Matrix4, 160
GetRow, 161
Matrix4, 160
operator∗, 161
Transpose, 161
SilverLining::MemObject, 162
SilverLining::MillisecondTimer, 164
∼MillisecondTimer, 165
GetMilliseconds, 165
MillisecondTimer, 165
SilverLining::Plane, 165
GetDistance, 168
GetNormal, 168
Normalize, 168
Plane, 167
SetDistance, 168
SetNormal, 168
SilverLining::RandomNumberGenerator, 168
∼RandomNumberGenerator, 169
SilverLining::ResourceLoader, 169
FreeResource, 171
GetFilesInDirectory, 171
LoadResource, 171
SetResourceDirPath, 172

SilverLining::Vector3, 172
Cross, 175
Dot, 175
Length, 175
Normalize, 175
operator∗, 175
operator+, 175
operator-, 176
operator==, 176
Serialize, 176
SquaredLength, 176
Unserialize, 176
Vector3, 174
x, 176
SilverLining::Vector3f, 177
Dot, 179
Length, 179
Normalize, 179
operator∗, 179
operator+, 179
operator-, 179
Vector3f, 178, 179
x, 180
SilverLining::Vector4, 180
Dot, 182
operator∗, 182
Vector4, 182
x, 182
SilverLining::Vertex, 182
SetColor, 184
SetUV, 184
SilverLining::WindVolume, 184
∼WindVolume, 187
GetDirection, 187
GetMaxAltitude, 187
GetMinAltitude, 187
GetWindSpeed, 187
Inside, 188
Serialize, 188
SetDirection, 188
SetMaxAltitude, 188
SetMinAltitude, 189
SetWindSpeed, 189
WindVolume, 187
SL_MAP
SilverLining::AtmosphericConditions, 107
SilverLining::CloudLayer, 128
SL_VECTOR
SilverLining::Atmosphere, 92
SquaredLength
SilverLining::Vector3, 176
STRATOCUMULUS
CloudTypes.h, 198
STRATUS
CloudTypes.h, 197

THA
LocalTime.h, 205
TimeZones
LocalTime.h, 204
ToGrayscale
SilverLining::Color, 135
Transpose
SilverLining::Matrix3, 157
SilverLining::Matrix4, 161

Unserialize
SilverLining::Atmosphere, 93
SilverLining::Color, 136
SilverLining::Vector3, 176
UpdateEphemeris
SilverLining::Atmosphere, 93
UpdateSkyAndClouds
SilverLining::Atmosphere, 93

Vector3
SilverLining::Vector3, 174
Vector3f
SilverLining::Vector3f, 178, 179
Vector4
SilverLining::Vector4, 182

WAT
LocalTime.h, 205
WindVolume
SilverLining::WindVolume, 187

x
SilverLining::Vector3, 176
SilverLining::Vector3f, 180
SilverLining::Vector4, 182

YST
LocalTime.h, 205